Comparison of Data Formats in Graph Neural Networks (GNN) and Their Computational Efficiency in PyTorch Geometric (PyG) and Deep Graph Library (DGL)

Patryk Graban
Gdańsk University of Technology
Faculty of Applied Physics and Mathematics

Błażej Kardynał
Gdańsk University of Technology
Faculty of Applied Physics and Mathematics

Bartosz Bycul
Gdańsk University of Technology
Faculty of Applied Physics and Mathematics

MSc Robert Benke
Project Supervisor
Gdańsk University of Technology
Department of Computer Systems Architecture

Abstract

This study focuses on the detailed analysis and comparison of data formats used in Graph Neural Networks (GNNs). The research is centered around two popular libraries: PyTorch Geometric (PyG) and Deep Graph Library (DGL).
The objective is to understand which data formats (such as CSR, COO, CSC) offer the highest computational efficiency in the context of various GNN models (Graph Convolutional Networks, GraphSAGE, Graph Attention Networks) and their applications. The primary goal of this project is to address questions regarding the impact of data formats on computation time.
We hypothesize that the choice of data format significantly affects computational efficiency and that differences in results will grow with the size of the input data. Additionally, tests conducted on GPUs will demonstrate greater efficiency compared to CPUs, and attention-based models (e.g., GAT) will exhibit higher computational overhead compared to simpler architectures, such as GCN.
The results of this study revealed significant performance differences depending on the applied data format and library. Performance analysis was conducted on datasets such as Planetoid (Cora, Citeseer, PubMed) and OGB (ogbn-arxiv). Key recommendations for choosing data formats and libraries based on applications were identified. Detailed results and analyses are presented in the following sections of this study.

Keywords:

Graph Neural Networks, PyTorch Geometric, Deep Graph Library, Sparse Data Formats, Computational Efficiency, Graph Convolutional Networks, GraphSAGE, Graph Attention Networks, Batching, CPU, GPU, Training Time

1. Objective

The aim of this study is a detailed analysis and comparison of data formats used in Graph Neural Networks (GNNs) and their impact on computational efficiency in the PyTorch Geometric (PyG) and Deep Graph Library (DGL) frameworks. The study aims to answer which data format (e.g., COO, CSR, CSC) provides the best performance in various computational scenarios. The research hypothesis assumes that the choice of an appropriate format can significantly impact computation time and overall efficiency of GNN models.

2. Outline and Theory

Graph Neural Networks (GNNs) enable efficient processing of graph-structured data, which finds broad applications in fields such as social networks, chemistry, and recommendation systems. A key feature of these models is their ability to capture relationships between nodes and propagate information within the graph, allowing the analysis of complex dependencies and data structures.

As the size of graphs grows, such as networks with millions of nodes and billions of edges, so do the computational power and memory requirements. Processing large graphs requires efficient data formats (e.g., CSR, COO) and optimized implementations on high-performance hardware, such as GPUs.

The scalability of GNNs largely depends on the optimization of data storage and processing methods, making the choice of an appropriate data format crucial for achieving high computational efficiency. In such contexts, analyzing various formats and their impact on model performance becomes especially important.

2.1. Core GNN Models

The most commonly used models within GNNs:

- **Graph Convolutional Networks (GCN)**: This model introduces convolutional operations on graphs, enabling information propagation between neighboring nodes. It utilizes adjacency matrix normalization and self-loops, which enhance numerical stability and computational efficiency. GCN is widely used in node classification and community detection tasks. Mathematically, the process can be expressed as:

$$H^{(k)} = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(k-1)}W^{(k)}\right), \tag{1}$$

  where $\tilde{A} = A + I$ is the adjacency matrix with added self-loops, $\tilde{D}$ is the degree matrix, $H^{(k)}$ is the feature matrix after the k-th iteration, $W^{(k)}$ is the weight matrix, and σ is the activation function (e.g., ReLU). This model effectively averages the features of neighboring nodes while preserving their structural properties.

- **GraphSAGE**: GraphSAGE aggregates the features of neighboring nodes to generate dynamic node representations. Thanks to various aggregation methods (e.g., summation, averaging), this model is highly scalable and used in analyzing large graphs. The process can be expressed as:

$$h_v^{(k)} = \sigma\left(W^{(k)} \cdot \text{AGGREGATE}\left(\{h_u^{(k-1)} : u \in \mathcal{N}(v)\}\right)\right), \tag{2}$$

  where $\mathcal{N}(v)$ denotes the neighborhood of node v. GraphSAGE enables dynamic representation construction, making it more scalable for large graphs.

- **Graph Attention Networks (GAT)**: The GAT model introduces an attention mechanism that allows differentiating the importance of neighboring nodes during information propagation. This provides greater flexibility in analyzing complex graph structures, such as molecules or transport networks. For each neighboring node u, the attention weight α_{vu} is calculated as:

$$\alpha_{vu} = \frac{\exp\left(\text{LeakyReLU}\left(a^T \cdot [Wh_v \parallel Wh_u]\right)\right)}{\sum_{k \in \mathcal{N}(v)} \exp\left(\text{LeakyReLU}\left(a^T \cdot [Wh_v \parallel Wh_k]\right)\right)}, \tag{3}$$

  where a is the attention weight vector, $\parallel$ denotes concatenation, and W is the weight matrix. The attention mechanism enables the model to differentiate the importance of neighboring nodes during information propagation.

2.2. Sparse Matrix Representations

Graphs are often represented in the form of adjacency matrices, which are typically sparse (i.e., containing many zero values). In practice, various sparse matrix formats are used, including:

- **CSR (Compressed Sparse Row)**: The CSR format is ideal for working with sparse matrices, especially when frequently operating on rows. In CSR, data is stored in three arrays: one holds all non-zero values, another stores the column indices of these values, and a third indicates the beginning of each row in these arrays. This format allows quick access to entire rows, making it highly efficient for matrix-vector multiplication. CSR saves memory but adding new elements can be challenging as it may require restructuring the entire data structure. This format is widely used in scientific computations.

- **CSC (Compressed Sparse Column)**: Analogous to CSR but optimized for column operations.

- **COO (Coordinate Format)**: The COO format is a simple and flexible way to store sparse matrices. Instead of a compact structure, it uses three arrays: one for row indices, one for column indices, and one for non-zero values. This format is convenient for building matrices piece by piece, as new elements can be easily added. However, since elements are unordered, arithmetic operations and matrix-vector multiplication are less efficient. COO is often used as an intermediate format for creating matrices before converting them into more efficient formats like CSR or CSC.

3. Practical Solutions

Graph Neural Networks (GNNs) are rapidly evolving, thanks to the introduction of new models and optimization techniques. One key achievement is the implementation of attention mechanisms, which allow dynamically differentiating the importance of information propagated between nodes. This mechanism, used in models such as Graph Attention Networks (GAT), increases flexibility and interpretability, enabling better modeling of relationships in complex networks, such as molecules or social networks[2].

In the context of scalability, one significant advancement is the GraphSAGE model, which introduces inductive representation learning. By sampling neighborhoods and aggregating node features, the model efficiently processes large graphs while reducing computational complexity[3].

Optimization of data representation is equally important. The COO format, a simple and flexible way to store sparse matrices, often serves as a transitional format. Advanced formats, such as CSR and CSC, significantly improve computational efficiency, especially in algebraic operations performed on rows and columns, respectively[1, 3]. CSR is widely used in analyzing large social graphs, while CSC is better suited for applications requiring intensive column operations[1, 2].

Together, these approaches enable more efficient and scalable graph data processing, which is crucial for applications like recommendation systems, molecular analysis, or social network optimization. Choosing appropriate data formats and models allows significant reductions in computational costs, particularly when working with large datasets[1, 2, 3].

4. Our Approach

Our approach focused on leveraging existing methods and techniques to compare the computational efficiency of data formats and the PyTorch Geometric (PyG) and Deep Graph Library (DGL) frameworks in the context of Graph Neural Networks (GNNs). We focused on analyzing three key time metrics:

- **Edge-to-edge time:** The total time from data loading, through weight computation, to forward pass.
- **Training time:** The time spent computing gradients.
- **Inference time:** The time for generating predictions from the trained model.

Experiments were conducted on datasets from the Planetoid family (*Cora*, *Citeseer*, *PubMed*) and OGB (*ogbn-arxiv*). To optimize computational resource utilization and accelerate the processing of large graphs, we applied batching, i.e., dividing input data into smaller portions (*batch size*).

Batching enables efficient utilization of GPU memory, particularly for very large datasets that cannot fit into memory all at once. This allows simultaneous processing of smaller subgraphs, reducing memory requirements and shortening training time.

5. Results and Analysis

5.1. PubMed Dataset

We began with the relatively small PubMed dataset from the Planetoid family, representing a citation network of biomedical research articles. It is widely used in GNN research. Tests were performed on the aforementioned models, considering the appropriate data formats and devices: CPU (central processing unit) and CUDA (Compute Unified Device Architecture), a universal architecture for multi-core processors, primarily GPUs (referred to as GPU in the following sections). The results were as follows.

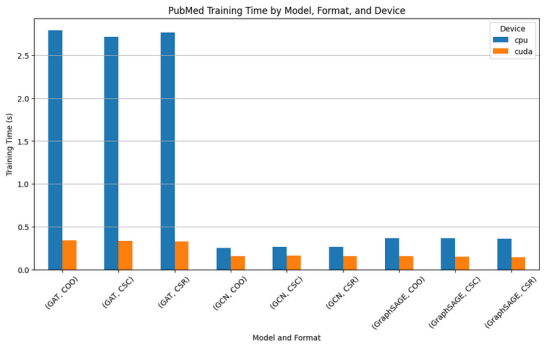

Figure 1: PubMed - Training Time, Batch 32

The GAT model exhibited the longest training time. This difference was particularly evident on the CPU, where training time was several times longer than that of GCN or GraphSAGE. On the GPU, the GAT model showed significant performance improvement, reducing training time by over 85% compared to the CPU.

GCN was the most efficient model on both CPU and GPU. Its training time was several times shorter than GAT and approximately 40% shorter than GraphSAGE on the GPU. GraphSAGE achieved intermediate results between GCN and GAT, showing stable training times on the GPU, where differences between data formats were marginal.

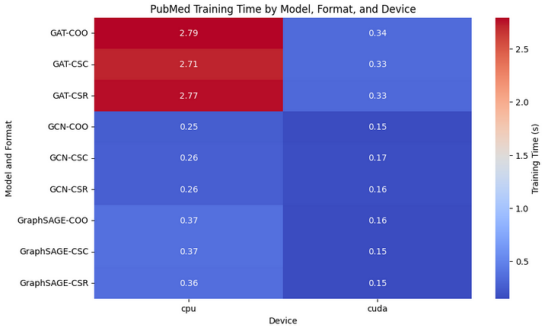

Figure 2: Heatmap of Training Time Dependencies

GPU vs. CPU Differences

Format	Model	CPU E2E Time (s)	CPU Training Time (s)	CPU Inference Time (s)	GPU E2E Time (s)	GPU Training Time (s)	GPU Inference Time (s)
COO	GAT	3.105264	2.768260	0.280342	0.447127	0.352298	0.038054
	GCN	0.326776	0.275998	0.029977	0.255254	0.193430	0.021584
	GraphSAGE	0.472690	0.398942	0.046801	0.252988	0.192258	0.015194
CSC	GAT	3.007289	2.698074	0.258392	0.426476	0.344486	0.033771
	GCN	0.351501	0.294031	0.033776	0.249097	0.186129	0.024745
	GraphSAGE	0.488774	0.414741	0.047038	0.213618	0.166814	0.010439
CSR	GAT	3.002700	2.668755	0.275163	0.430550	0.344484	0.036695
	GCN	0.321865	0.273939	0.029932	0.237027	0.185154	0.019376
	GraphSAGE	0.459006	0.393714	0.044394	0.241117	0.180184	0.015260

Figure 3: Detailed Data for Batch 32

Format	Model	CPU E2E Time (s)	CPU Training Time (s)	CPU Inference Time (s)	GPU E2E Time (s)	GPU Training Time (s)	GPU Inference Time (s)
COO	GAT	3.111138	2.818688	0.250813	0.432778	0.324389	0.043409
	GCN	0.281645	0.232906	0.029800	0.177470	0.115755	0.022407
	GraphSAGE	0.410397	0.336761	0.046538	0.171694	0.121674	0.011201
CSC	GAT	3.033298	2.731669	0.257454	0.427617	0.324879	0.041849
	GCN	0.282598	0.234318	0.029874	0.205581	0.144891	0.022362
	GraphSAGE	0.385038	0.320707	0.043832	0.199795	0.137466	0.015113
CSR	GAT	3.173167	2.866448	0.259039	0.419628	0.317967	0.043379
	GCN	0.300560	0.252053	0.030122	0.191065	0.133628	0.021275
	GraphSAGE	0.390162	0.325056	0.044214	0.170262	0.114835	0.013767

Figure 4: Detailed Data for Batch 64

For the GAT model, the differences in training times between GPU and CPU were the largest, exceeding 90% for both batch sizes and all data formats. For GCN and GraphSAGE, GPU-CPU differences were less pronounced but still exceeded 60%, highlighting the benefits of GPU acceleration for these models.

Impact of Batch Size

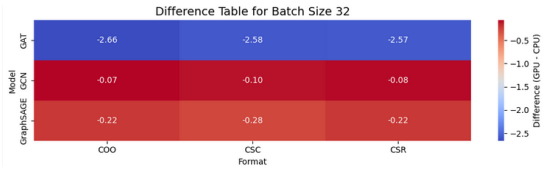

Figure 5: GPU Difference (Batch 32)

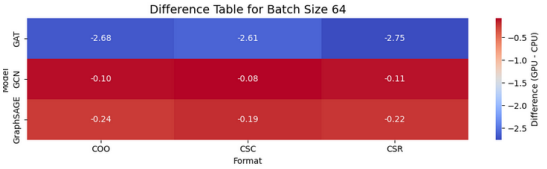

Figure 6: GPU Difference (Batch 64)

Increasing the batch size from 32 to 64 positively affected GPU training times, reducing them by an average of 15% for all models. On the CPU, the impact of batch size was less noticeable, with minor changes in training times (below 5%). The greatest benefits of a larger batch size were observed for GCN and GraphSAGE, where time reductions were most evident.

Data Formats

CSR and CSC formats showed very similar results in most cases, especially on the GPU, where training time differences were below 5% for all models. The COO format was noticeably slower on the CPU, particularly for the GAT model, where the difference compared to CSR and CSC exceeded 10%. On the GPU, these differences were less significant, remaining below 3%.

5.2. Cora Dataset

Cora is one of the most popular datasets, also part of the Planetoid family. It represents a citation network of machine learning research articles.

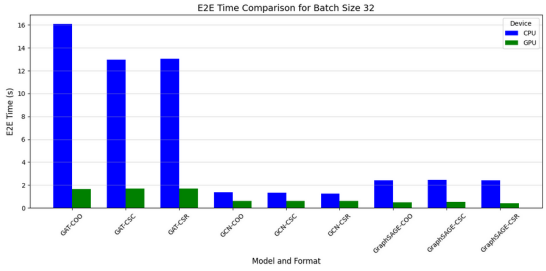

Figure 7: Cora Dataset - Training Time, Batch 32

Datasets from the Planetoid family are relatively similar, so their analysis is expected to reveal similar observations. Specifically, the GAT model showed the highest

training times on the CPU compared to GCN and GraphSAGE, just like in the PubMed dataset. Differences were particularly noticeable for smaller batch sizes, where the training time on the CPU was several times longer than GCN.

GCN, as the most efficient model, achieved training times over 80% shorter than GAT on the CPU and approximately 50% shorter on the GPU. This is naturally related to the reduced number of operations required by this model.

GraphSAGE was positioned between GCN and GAT in terms of performance. Training times on the GPU for GraphSAGE were stable and differed slightly between data formats, indicating its good optimization.

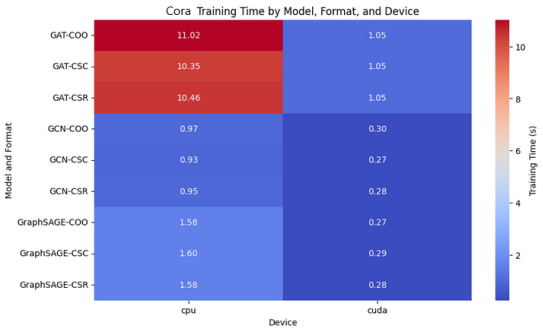

Figure 8: Heatmap of Cora Dataset - Average Across Batches

GPU Utilization

The greatest benefits of GPU usage were observed for the GAT model, where training time on the GPU was over 90% shorter compared to the CPU. For GCN and GraphSAGE, differences were smaller but still significant, exceeding 70% in most cases (see heatmaps and bar charts for details).

Batching

With increasing batch size, improvements in GPU efficiency were observed. Training times decreased by an average of 20% when transitioning from a batch size of 32 to 128 for all models. On the CPU, the effect of batch size was less pronounced, but increasing the batch size to 128 led to stabilized training times, particularly for GCN and GraphSAGE.

Cora Dataset - CPU Format and Model E2E Time Comparison by Batch Size:

	E2E Time (s)								
Format	COO			CSC			CSR		
Model	GAT	GCN	GraphSAGE	GAT	GCN	GraphSAGE	GAT	GCN	GraphSAGE
Batch Size									
32	16.083444	1.390213	2.395665	12.987549	1.333699	2.458369	13.061496	1.246826	2.431277
64	11.372219	1.094461	1.739607	11.196409	1.019279	1.709191	11.204181	1.043790	1.726140
128	9.899184	0.890914	1.385452	9.560788	0.840513	1.453852	9.747180	0.907361	1.378708

Figure 9: Cora Dataset - CPU Batch Size Dependency

Cora Dataset - GPU Format and Model E2E Time Comparison by Batch Size:

	E2E Time (s)								
Format	COO			CSC			CSR		
Model	GAT	GCN	GraphSAGE	GAT	GCN	GraphSAGE	GAT	GCN	GraphSAGE
Batch Size									
32	1.659554	0.605574	0.470627	1.710260	0.617266	0.524351	1.689765	0.618111	0.421710
64	1.294106	0.324861	0.345566	1.288344	0.313979	0.360696	1.292594	0.311449	0.388084
128	0.921137	0.315045	0.266001	0.908852	0.203561	0.278805	0.903066	0.235095	0.255244

Figure 10: Cora Dataset - GPU Batch Size Dependency

Data Format Performance

On the CPU, the GAT model performed worse on the COO matrix compared to other formats. CSR and CSC formats, however, yielded similar results. On the GPU, these results were less consistent, with differences between formats becoming negligible in some cases. This suggests that the importance of the format becomes more apparent with computationally demanding models.

5.3. ArXiv Dataset

We now turn to a significantly larger dataset from the Open Graph Benchmark (OGB), featuring 169,000 nodes and 1.1 million edges. It represents a citation network of research articles across multiple fields published on the ArXiv platform.

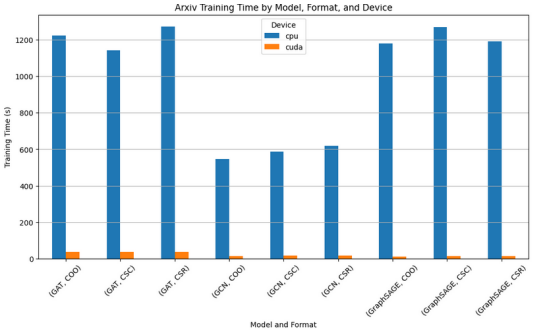

Figure 11: ArXiv Dataset - Aggregated Results

In some cases, training time on the CPU was over 10 times longer than on the GPU. This was most noticeable for the GraphSAGE model.

Interestingly, for the CPU, GraphSAGE showed similarities to GAT, contrary to what was observed in smaller datasets.

GCN was the most efficient model, achieving training times over 80% shorter than GAT on the CPU and approximately 50% shorter on the GPU.

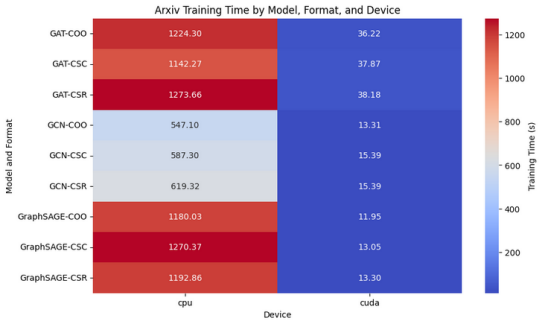

Figure 12: ArXiv Dataset - Heatmap of Batch Size Effects

Batch Size Effects

Increasing the batch size from 32 to 64 reduced training times on both GPU and CPU, maintaining the trends observed in smaller datasets. However, the scale of this reduction was significantly larger for this dataset, highlighting the importance of GPU usage for processing such large datasets.

Format	Model	CPU E2E Time (s)	CPU Training Time (s)	CPU Inference Time (s)	GPU E2E Time (s)	GPU Training Time (s)	GPU Inference Time (s)
COO	GAT	1543.727375	1440.632495	101.113215	41.111250	36.743870	2.609728
	GCN	906.740286	755.795412	148.086487	22.073641	19.906387	0.584578
	GraphSAGE	1854.048564	1541.934928	310.336894	20.620213	17.971669	0.797827
CSC	GAT	1380.391637	1282.035447	96.164650	45.737837	40.639152	2.830617
	GCN	906.121105	742.981165	161.208298	25.936280	22.185282	1.348098
	GraphSAGE	2060.747300	1734.870655	322.617797	21.539405	18.498070	0.807790
CSR	GAT	1484.254823	1381.568479	100.460989	45.538499	40.503433	2.785275
	GCN	917.625242	763.823670	151.944047	25.300709	22.482125	0.969545
	GraphSAGE	1887.303048	1564.935930	320.362062	21.600745	18.567652	0.806736

Figure 13: ArXiv Dataset - Numerical Results for Batch 32

Format	Model	CPU E2E Time (s)	CPU Training Time (s)	CPU Inference Time (s)	GPU E2E Time (s)	GPU Training Time (s)	GPU Inference Time (s)
COO	GAT	1064.213782	1007.965363	54.314518	40.646023	35.954262	2.825716
	GCN	558.166839	442.757336	113.596878	13.342774	10.012565	1.352334
	GraphSAGE	1112.812263	818.132538	292.746288	11.623813	8.931710	0.809108
CSC	GAT	1066.310538	1002.503645	61.620951	39.907651	36.482784	2.052710
	GCN	552.125175	431.619170	118.544619	15.087394	11.987372	1.110366
	GraphSAGE	1104.130486	805.864011	296.457507	12.807548	10.332416	0.594350
CSR	GAT	1218.985236	1165.748724	51.038721	42.099890	37.021596	2.831964
	GCN	589.390741	474.809863	112.779051	15.391453	11.851060	1.345041
	GraphSAGE	1127.780205	820.779196	304.969949	13.322483	10.658722	0.682729

Figure 14: ArXiv Dataset - Numerical Results for Batch 64

Data Format Performance

The study on the ArXiv dataset does not allow for definitive conclusions about format characteristics compared to others. An observed anomaly was the COO format being the fastest on the GPU, which contradicts the initial hypothesis. This requires further investigation, focusing exclusively on this aspect.

6. Conclusion

GPU Efficiency vs. CPU
Across all datasets, GPU usage led to significant reductions in training and inference times. These benefits were more apparent for larger datasets, such as PubMed and ArXiv, where time differences between GPU and CPU exceeded 90%. For smaller datasets, such as Cora, GPU also brought advantages, but the performance differences were less pronounced due to lower computational demands.

Impact of Batch Size
Larger batch sizes positively impacted GPU efficiency, reducing training times by an average of 15–25% when increasing from 32 to 128. On the CPU, the effect was less significant but noticeable in terms of stabilizing computation times.

Model Characteristics

- **GCN**: The most efficient model on both CPU and GPU across all datasets, attributed to its simplicity and lower computational demands.
- **GraphSAGE**: Stable performance, particularly on the GPU. This model demonstrated good scalability, especially for large datasets.
- **GAT**: The most computationally demanding model with the highest training times, especially on the CPU. However, it showed the greatest benefits from GPU usage, reducing training times by over 85% compared to the CPU.

Data Format Impact

- CSR and CSC formats offered comparable performance, particularly on the GPU, where differences between these formats were marginal.
- COO format exhibited greater variability in results, sometimes achieving performance similar to CSR and CSC, but in other cases proving less efficient, especially on the CPU. For large datasets, comparisons yielded inconclusive results, motivating further research in this area.

Key Findings

- **GPU Optimization**: GPU demonstrates significant advantages in GNN tasks, particularly for more complex models (e.g., GAT) and large datasets.
- **Data Formats**: CSR and CSC are the most efficient in analyzed scenarios, but further optimization of COO may increase its utility in specific cases.
- **Batch Size Application**: Larger batch sizes improve GPU efficiency, which is critical for processing large graphs.

Overall, the choice of model, data format, and parameters such as batch size is crucial for computational efficiency in GNN applications. Further exploration of algorithm structures will enable not only optimization but also the development of new methods to significantly enhance efficiency in graph applications.

References

[1] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *arXiv preprint arXiv:1706.02216*, 2017.

[2] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2017.

[3] Petar Veličković et al. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.