Enhancing Model Utility of Differentially Private Graph Neural Networks

Anirban Dasgupta [1]   Abhinav Khot [1]   Arjun B Dikshit [1]

[1]Indian Institute of Technology Gandhinagar

Introduction

Differential Privacy

Differential Privacy quantifies the protection of data in a mathematical manner. It models the effect that one training point in a dataset has on the model outputs.More formally, a randomized algorithm $\mathcal{M}$ with domain $\mathbb{N}^{|\mathcal{X}|}$ is (ε, δ)-differentially private if for all $\mathcal{S} \subseteq \text{Range}(\mathcal{M})$ and for all $x, y \in \mathbb{N}^{|\mathcal{X}|}$ such that $\|x - y\|_1 \leq 1$:

$$\Pr[\mathcal{M}(x) \in \mathcal{S}] \leq e^{\varepsilon} \cdot \Pr[\mathcal{M}(y) \in \mathcal{S}] + \delta \tag{1}$$

Here, $\mathbb{N}^{|\mathcal{X}|}$ represents the space of all possible datasets, where $\mathcal{X}$ is the data universe. The notation $\|x - y\|_1 \leq 1$ indicates that datasets x and y differ in at most one element (i.e., one is a proper subset of the other with one additional or one fewer element). For Graphs, node level differential privacy involves datasets differing by a node, its associated edges from the dataset.

The most common way achieve Differential privacy is through the addition of calibrated noise. Noise can perturb the probability distributions to satisfy the guarantee .DP-SGD is a modification of standard stochastic gradient descent that incorporates differential privacy. In each iteration, a random mini batch is sampled from the dataset, and individual gradients are computed for each data point. These gradients are then clipped to a fixed norm C to limit the influence of any single example. After clipping, Gaussian noise scaled by σC is added to the sum of the gradients to ensure privacy and then this quantity is divided by batch size. The model parameters are then updated using this noisy, clipped average gradient. This procedure provides a formal differential privacy guarantee while enabling the training of effective machine learning models.

Model Overview & Problem Formulation

We work with **DPAR**(Decoupled GNN with Differentially Private Approximate Personalized PageRank), introduced by Zhang et.al 2018.

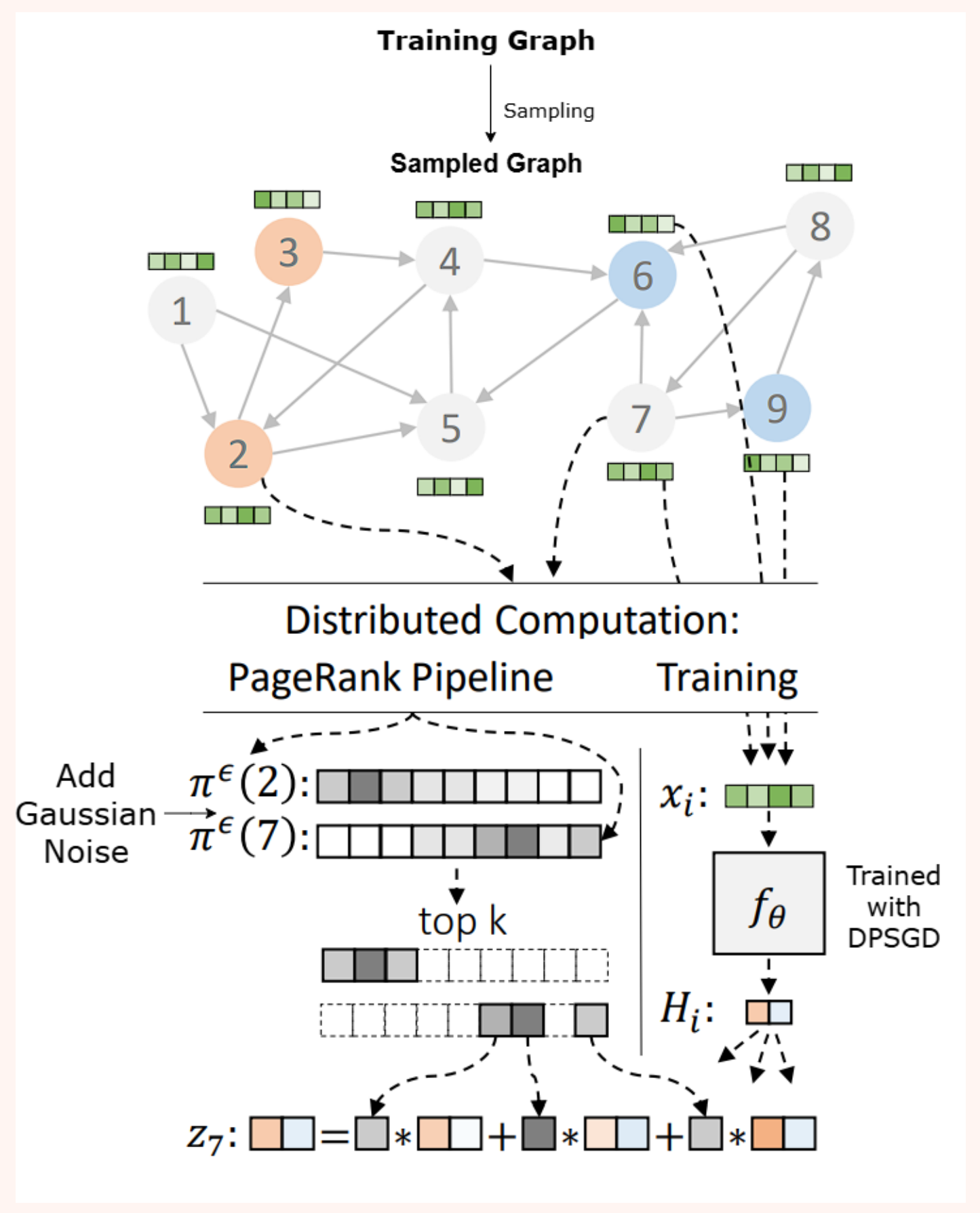


Figure 1. Illustration of the DPAR architecture

One pipeline calculates Personalized PageRank for the nodes of the graph whereas the other encodes the node features into a low dimensional representation. For each node, the top-k nodes as determined by the Personalized PageRank vector are used for aggregating the low dimensional representations to get the final logits which are used for classification using softmax. We experiment with modifying various components of this model to enhance and understand model utility. We also test the model against privacy attacks to understand what part of the model provides protection.

Modifying the Personalized PageRank Algorithm

We experiment with alternatives to the traditional Personalized PageRank Algorithms.

- Feature aware Random-Walk : In the regular Personalized PageRank algorithm, every edge originating from a node is given the same transition probability. We modify this to give the weight based on cosine similarity between the nodes incident on the edge.
- Clustering Method: We cluster the nodes and take the "centroid" of the cluster to be the average of node features in the cluster. For the purpose of aggregation, we use nearest top-k such centroids.

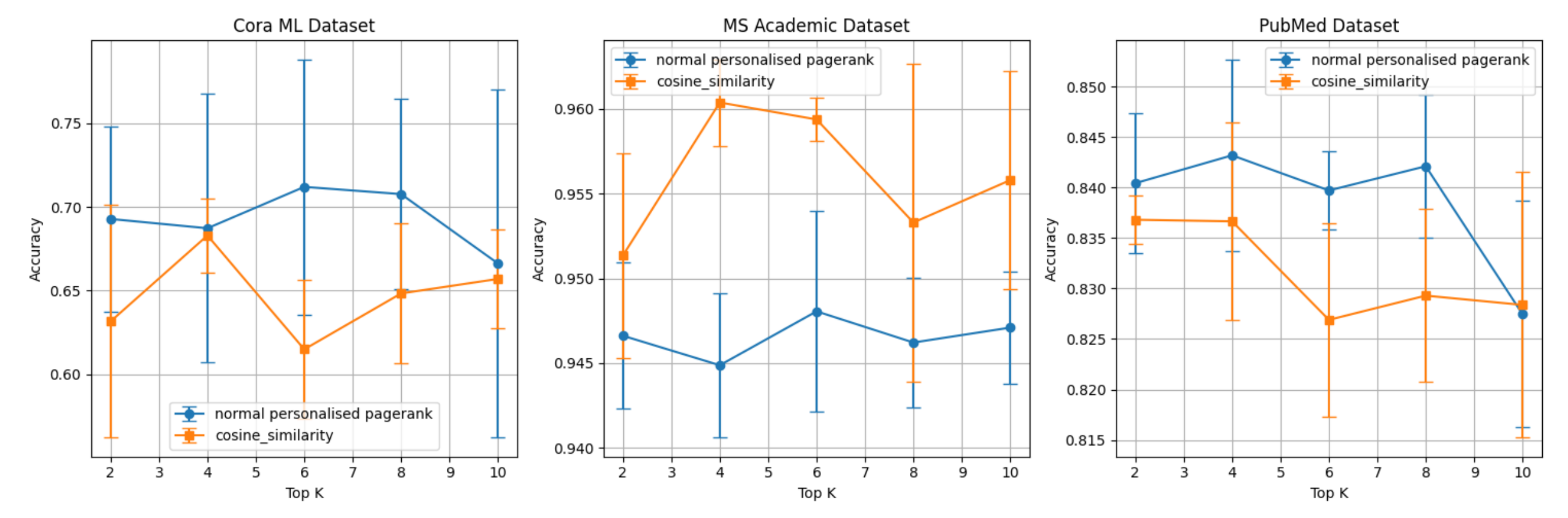
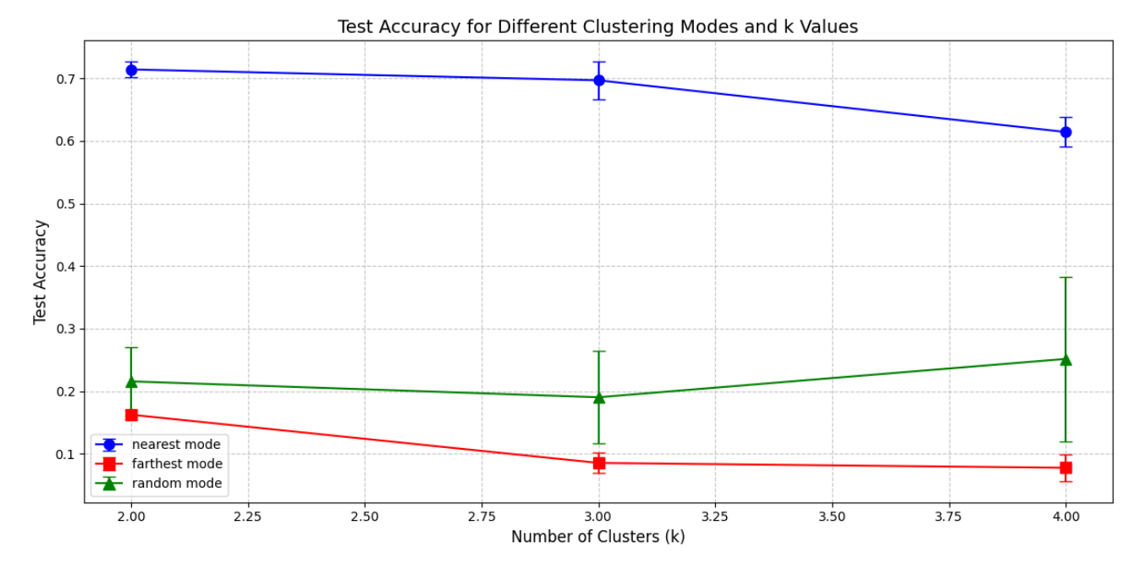
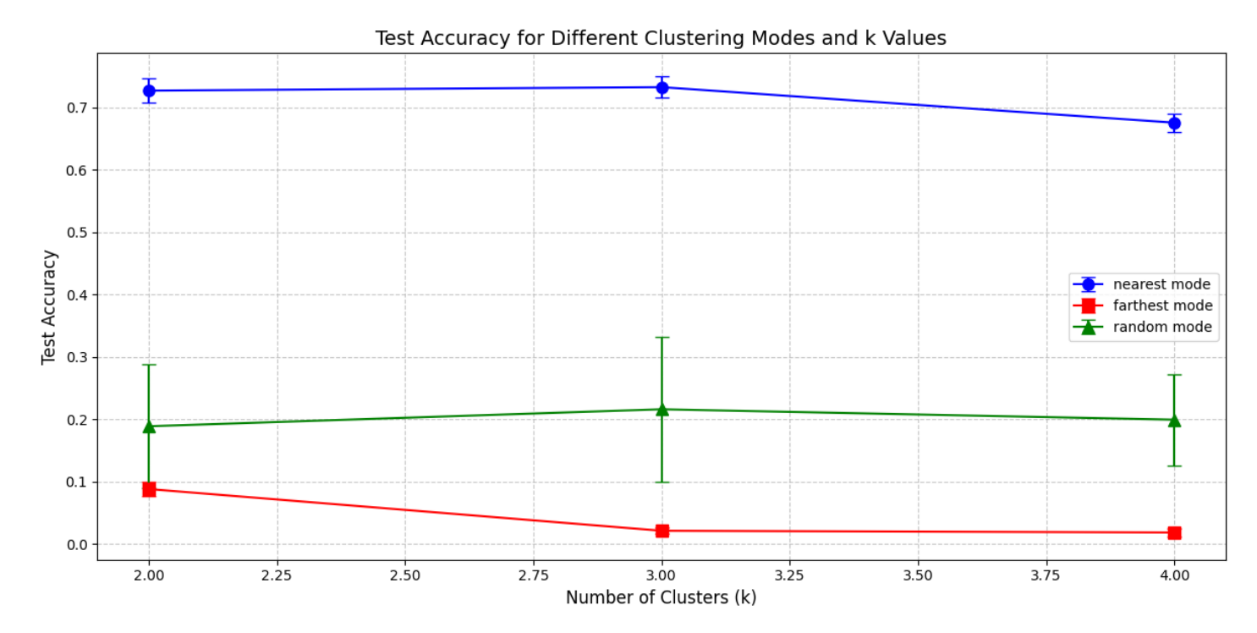


Figure 2. Feature Aware Random Walk. DPAR trained on 9% sampled graph. Non private setting

The feature aware random walk based PPR works almost as well regular PPR with a nominal improvement only in the MS Academic Dataset. Differential private settings show similar results too.



(a) Clustering Method. Euclidean distance used as metric.



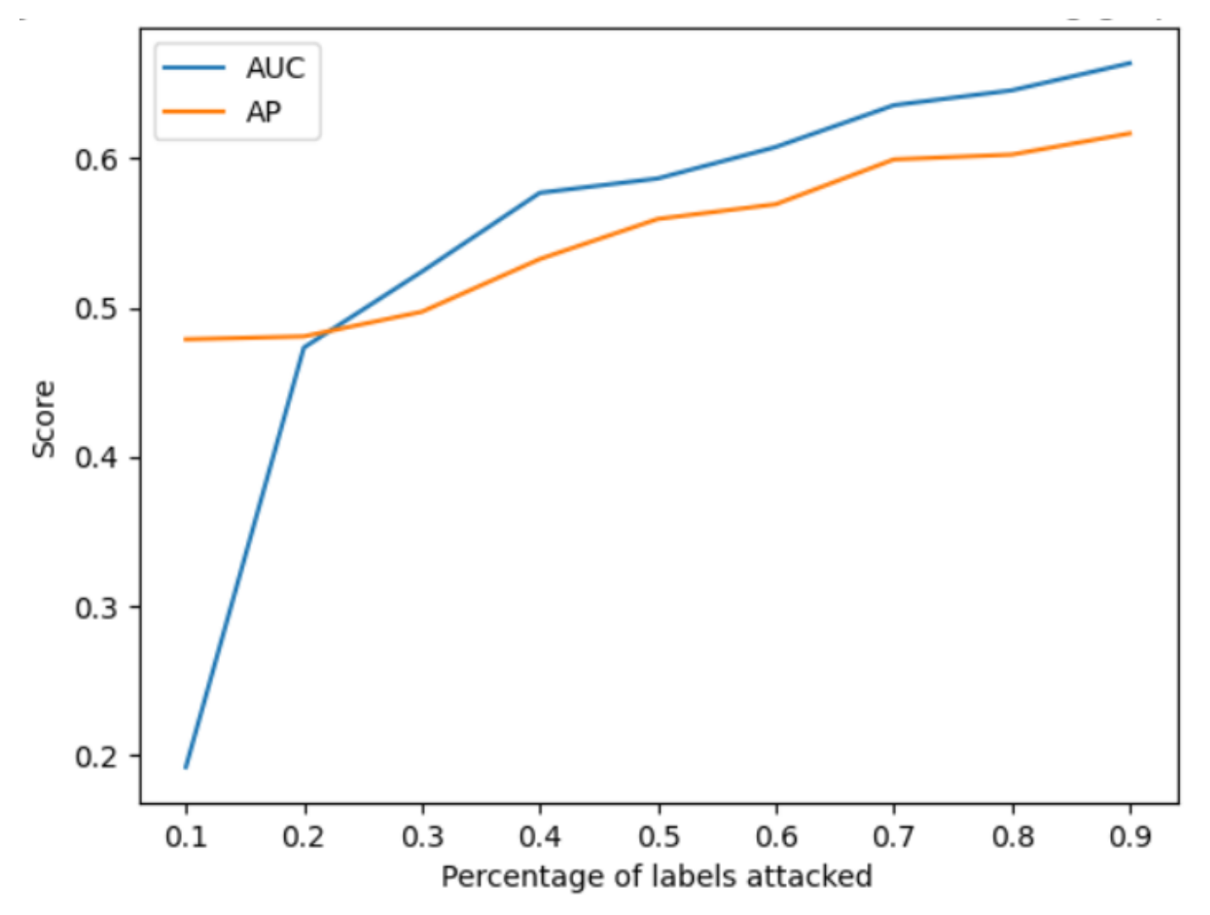
(b) Clustering method. Cosine Similarity used as metric.

Figure 3. Comparison of clustering methods (CoraML Dataset). Graph not sampled for training

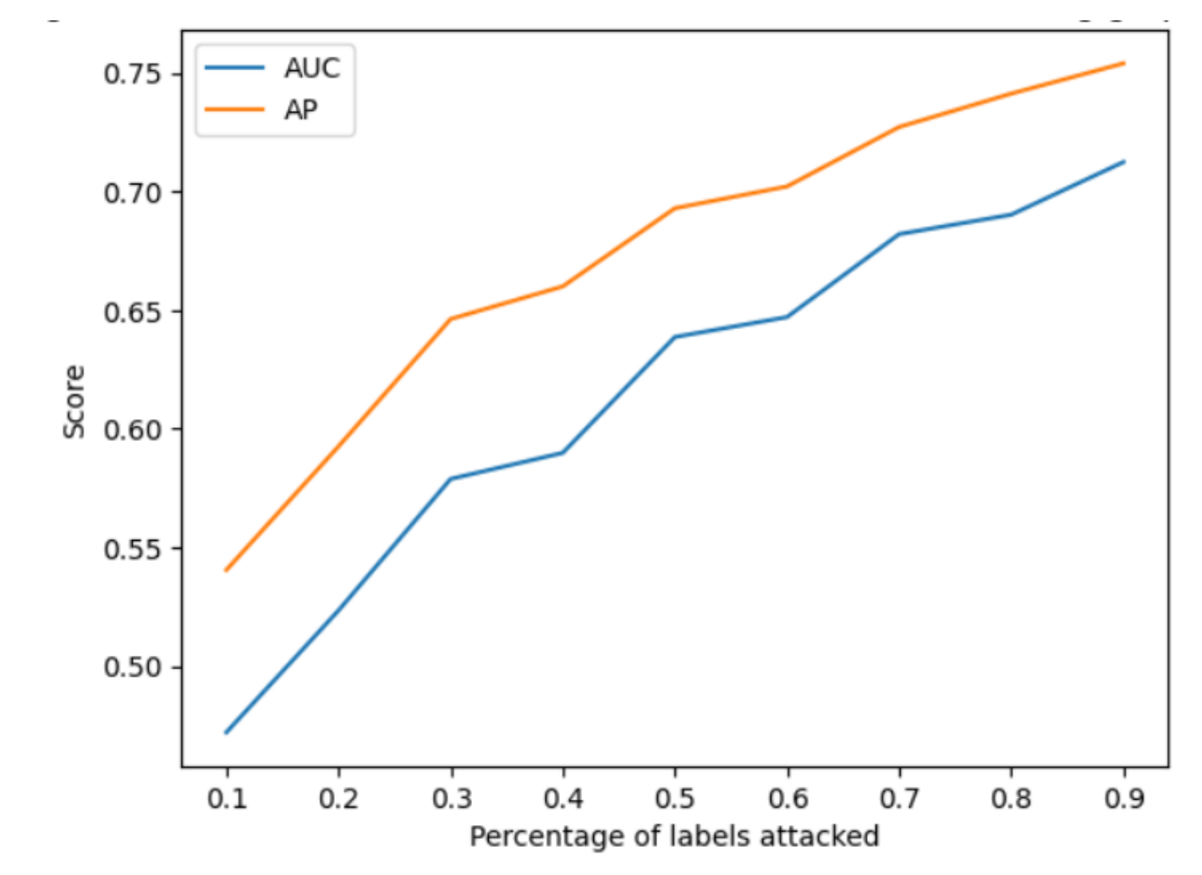
From these results, we observe that aggregation with similar centers is always better and although counter - intuitive, aggregation with lesser cluster centers is beneficial. This arises due to the fact that on increasing k, we are adding a center which is unrelated to the representation and can skew results.

Testing DPAR Against Model Inversion Attack

We use GraphMI, a projected gradient descent based attack which reconstructs the training adjacency matrix from the node features and a subset of the node labels. Since this attack needs the node features to be know a priori, we train DPAR only with DP-APPR for the upcoming experiments.



(a) Inversion Attack with DPAR trained on 50 % sampled graph

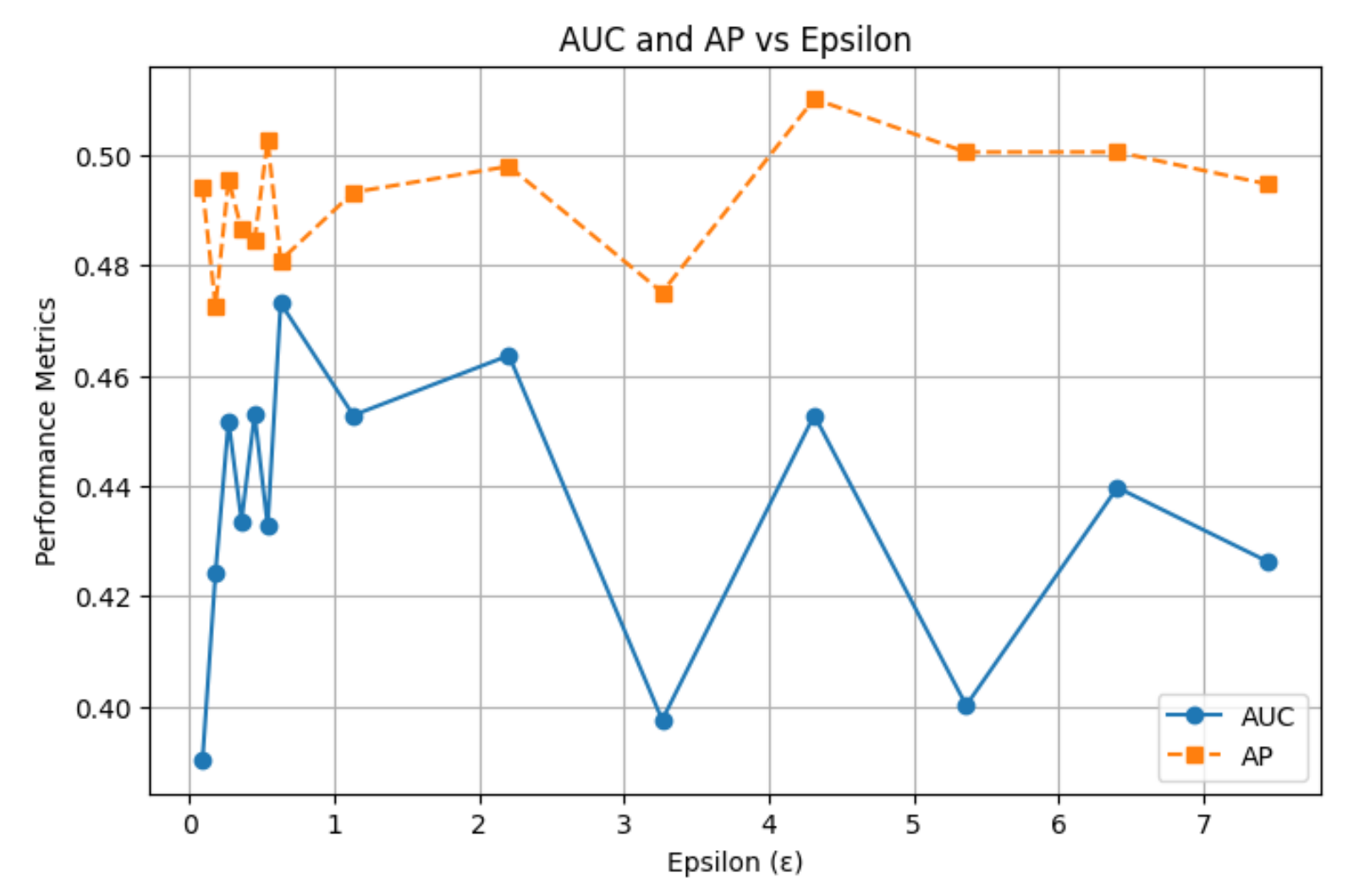


(b) Inversion Attack with DPAR trained on entire graph

Figure 4. Comparison of DPARs performance under different attack settings

We see that sampling the training graph is a huge deterrent to the model inversion attack. From the proceeding image, we observe that varying our privacy budget (epsilon) has relatively little effect on the attack scores, indicating that DP-APPR in DPAR is not a good defense against MI attacks. However, even 50% sampling drops the attack score by 13% even in the worst case scenario of all the node labels being known, proving that sampling is a good defense.

We tried to rewire the edges and (separately) drop random edges in the original graph used to train DPAR, as suggested by the GraphMI paper to produce a defense against Model Inversion Attack. We noticed that doing so, did not affect the AUC and AP scores of the MI attack, nor did it affect the accuracy of DPAR by much.



(a)

% of Graph Sampled	AP Score	AUC Score
10	64.80 %	67.96%
50	63.30%	67.35%
100	76.14%	72.30%

(b)

Figure 5. Effect of Varying Privacy budget of DPAR (left) and Effect of Sampling graph (right) against Model Inversion Attacks(100 % labels attacked).

Alternative Graph Sampling methods

Since we observe that sampling is the only main deterrent to the model inversion attack, we try different graph sampling methods in order to improve the model utility.

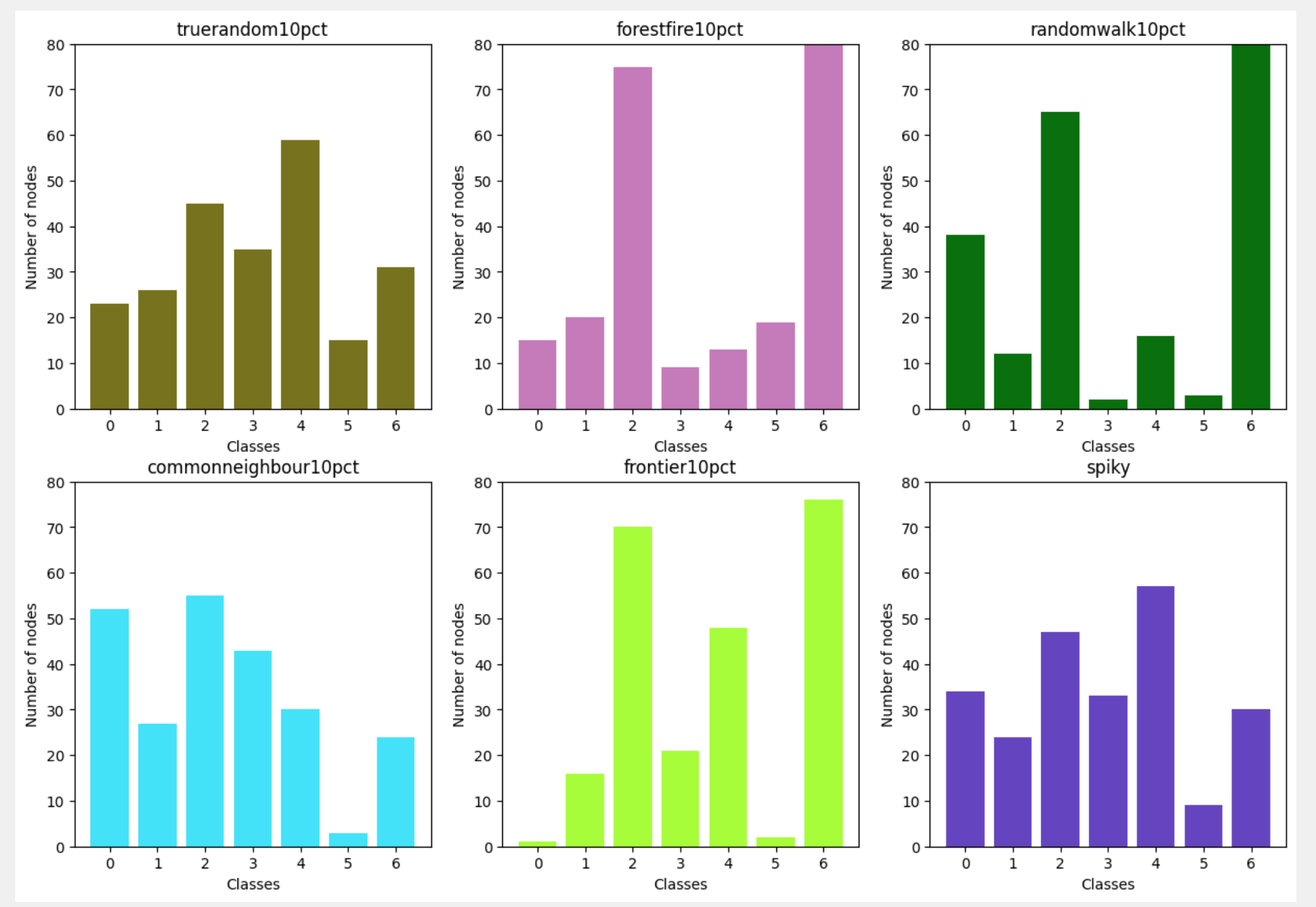


Figure 6. Histogram of the graph sampled with different methods

Forest Fire and Random Walk are methods that provide a highly connected graph structure from the node they started from, but may lead to class imbalance. The other methods explore more of the graph. We see that sampling methods which provide a better distribution over classes perform better. Thus, we try uniformly sampling nodes from each class.

Sampling Method	topk-2	topk-4	topk-6	topk-8
True Random	0.7889	0.7876	0.7901	0.7920
Spiky Ball	0.6223	0.5975	0.6347	0.6161
Frontier	0.6192	0.5573	0.5418	0.5604
Common Neighbour Aware	0.4737	0.5325	0.4799	0.5170
Random Walk	0.2539	0.2353	0.2384	0.2693
Forest Fire	0.2246	0.2786	0.2724	0.2570

Figure 7. Performance of different sampling methods across various top-k values.

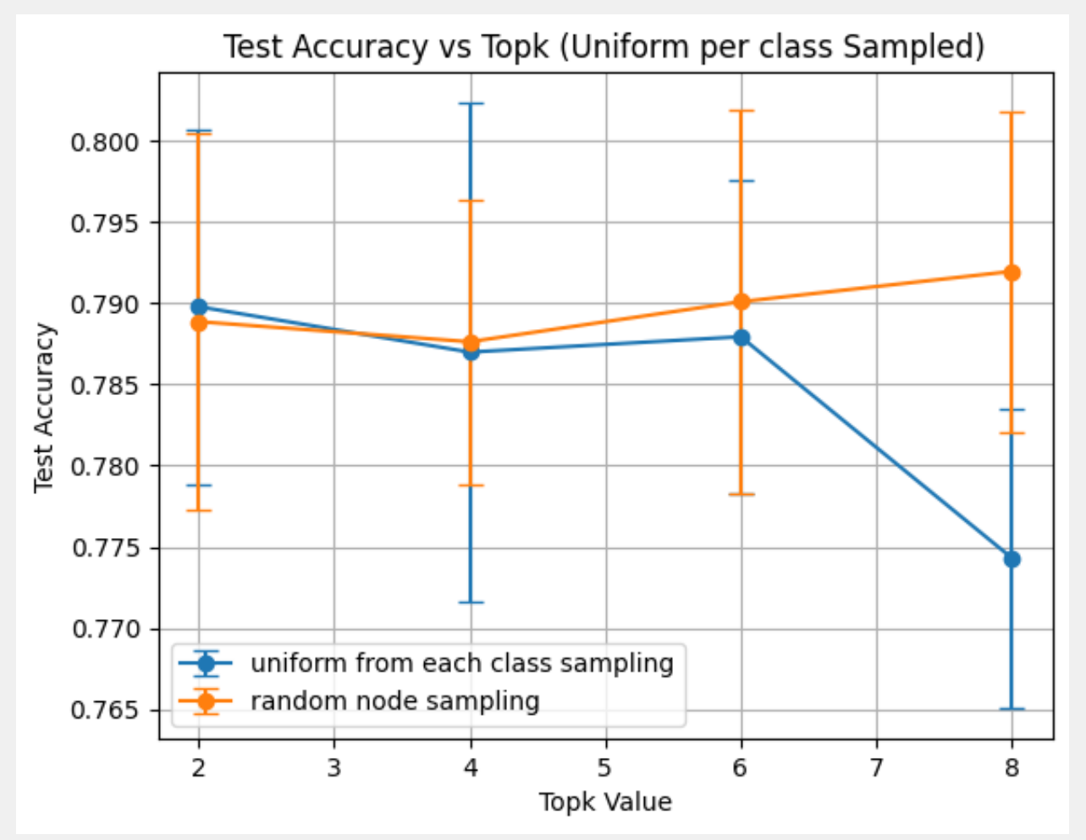


Figure 8. Sampling uniformly from each class.

We see very similar results to random sampling with a nominal 2 % improvement for topk-8.

References

[1] Aleksandar Bojchevski, Johannes Gasteiger, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann.
Scaling graph neural networks with approximate pagerank, August 2020.

[2] Benedek Rozemberczki, Oliver Kiss, and Rik Sarkar.
Little ball of fur: A python library for graph sampling, 2020.

[3] Qiuchen Zhang, Hong kyu Lee, Jing Ma, Jian Lou, Carl Yang, and Li Xiong.
Dpar: Decoupled graph neural networks with node-level differential privacy, May 2024.

[4] Zaixi Zhang, Qi Liu, Zhenya Huang, Hao Wang, Chee-Kong Lee, and Enhong Chen.
Model inversion attacks against graph neural networks, 2022.