Investigation of Pooling Mechanisms in Graph Neural Networks

Saatwik Patnaik (22B1284) Anilesh Bansal (22B0928) Chaitanya Garg (22B0979)
Utkarsh Pant (22B0914) Harsh Kumar (22B0973)

May 6, 2025

Abstract

This report presents a comprehensive empirical study of diverse graph pooling mechanisms in Graph Neural Networks(GNNs). We evaluate their classification accuracy, memory usage, and training time; conduct a robustness analysis under structural and feature perturbations; and perform a grid search to identify optimal pooling ratios across multiple benchmark datasets.

1 Introduction

Pooling mechanisms in GNNs play a role similar to that in standard neural networks. They reduce the size and complexity of feature representations while retaining the important and relevant information.
In the context of graphs, it helps in dimensionality reduction by compressing it into a lower dimensional form that helps in making later tasks like graph classification more computationally tractable. Additionally, it also supports hierarchial representation by coarsening the graph thereby allowing the network to learn both local and global patterns in the graph.
Selecting the right pooling strategy thus depends on graph size, task complexity and computational budget. We systematically investigate twelve pooling strategies on MUTAG, PROTEINS, and ENZYMES datasets with the goal to:

- Compare accuracy, runtime, and memory footprint.
- Analyze robustness to structural and feature perturbations.
- Identify optimal pooling ratios via grid search.

2 Methodology

2.1 Summary of the Pooling Mechanisms Implemented

We implement the following pooling methods:

- **no-pool**: No pooling; use $\{\mathbf{h}_v\}$ directly.
- **global-max**: $\mathbf{h}_G = \max_v \mathbf{h}_v$ (element-wise).
- **global-mean**: $\mathbf{h}_G = \frac{1}{|V|}\sum_v \mathbf{h}_v$ (averaging).
- **global-sum**: $\mathbf{h}_G = \sum_v \mathbf{h}_v$ (size-sensitive).
- **global-att**: $\mathbf{h}_G = \sum_v \alpha_v \mathbf{h}_v$, $\alpha_v \propto \exp(\mathbf{a}^\top \tanh(W\mathbf{h}_v))$.
- **set2set**: Recurrent attention pooling: $\mathbf{h}_G = [r_T; q_T]$.
- **topk**: Score $y_v = \mathbf{p}^\top \mathbf{h}_v$, keep top-$k$ nodes.
- **sag**: Like TopK but scores via GNN-attention; reweights survivors.
- **ecpool**: Edge score $s_{uv} = \mathbf{w}^\top[\mathbf{h}_u \parallel \mathbf{h}_v]$, merge top-$\rho|E|$.
- **dmon**: Soft assignment S with modularity loss $\mathcal{L}_{\text{mod}} = -\frac{\text{tr}(S^\top AS)}{2m}$.

- **mincut**: Assignment S with $\mathcal{L}_{\text{mincut}} = -\frac{\text{tr}(S^\top AS)}{\text{tr}(S^\top DS)}$.
- **diff**: Differentiable $S = \text{softmax}(\text{GNN}(X, A))$, then X', A' via S.

2.2 Experimental Setup

Datasets MUTAG, PROTEINS, ENZYMES via PyTorch Geometric's `TUDataset`. 80/20 train/test split.

Model 1-layer GCN backbone (hidden dim 64), followed by one pooling layer, then GCN + global mean pooling, and MLP classifier (in most cases).

Training 100 epochs, learning rate 0.01, batch size 32.

Metrics

- *Accuracy (%)* on test split.
- *Time per epoch (s).*
- *Peak GPU memory (MB)* via `torch.cuda.max_memory_allocated()`.

Robustness We apply:

- random edge dropout: np.linspace(0.0, 0.3, 16)
- random node dropout: np.linspace(0.0, 0.3, 16)
- Gaussian feature noise: np.linspace(0.0, 0.5, 16) as standard deviation of added noise

and measure accuracy degradation on MUTAG and PROTEINS datasets.

Grid Search We apply pooling ratios $r \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$ for SAGPool and EdgePool in MUTAG, PROTEINS and ENZYMES datasets and compare.

3 Results

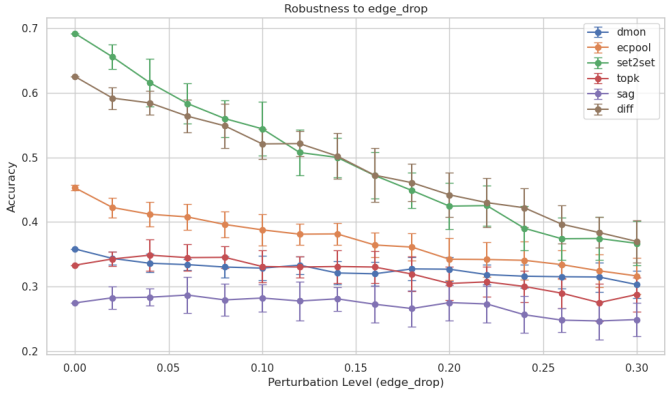
3.1 Pooling Comparison

Table 1: Performance of pooling methods on MUTAG, PROTEINS, and ENZYMES. Best accuracy and lowest time per dataset are bolded.

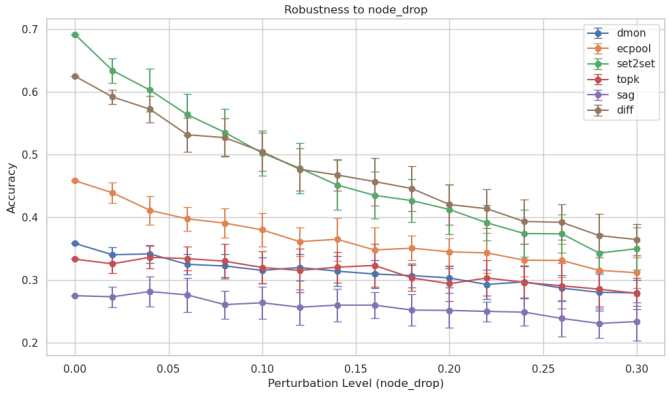
Method	MUTAG			PROTEINS			ENZYMES		
	Acc.	Time	Mem	Acc.	Time	Mem	Acc.	Time	Mem
dmon	84.2	0.05	18.7	**73.1**	0.26	78.6	29.2	0.13	23.0
ecpool	79.0	0.14	18.7	72.7	1.88	28.1	35.8	0.89	22.4
mincut	81.6	0.05	18.9	67.3	0.30	127.2	30.8	0.15	25.5
global-att	73.7	0.04	18.1	72.2	0.26	25.9	28.3	0.13	21.6
set2set	**86.8**	0.11	27.1	69.5	0.60	30.0	**50.8**	0.32	28.3
global-mean	71.1	**0.03**	18.8	70.8	**0.15**	26.6	34.2	**0.08**	22.3
global-sum	84.2	**0.03**	18.8	71.3	**0.15**	26.7	30.0	**0.08**	22.3
topk	76.3	0.04	18.8	71.3	0.25	25.6	26.7	0.13	21.9
sag	78.9	0.05	19.0	69.5	0.32	25.9	31.7	0.15	22.0
diff	**86.8**	0.04	19.9	**74.0**	0.31	78.1	44.2	0.15	24.0

3.2 Robustness Analysis

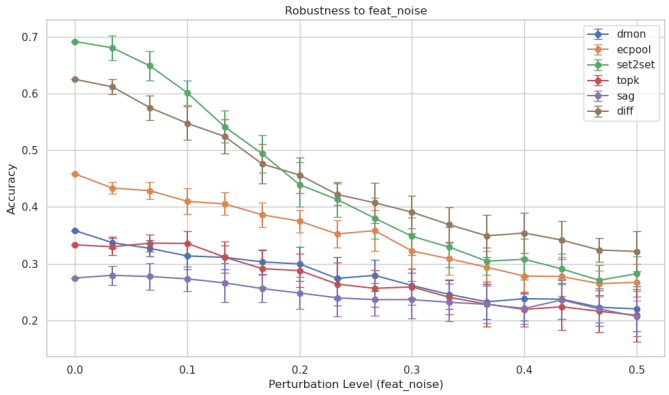
Below are the results obtained for adding edge, node and feature noise perturbations to different pooling mechanisms for ENZYMES dataset



(a) Edge dropping



(b) Node dropping



(c) Feature noise

Figure 1: Model robustness on the ENZYMES dataset under different perturbations

3.3 Pooling Ratio Grid Search

We performed gridsearch on different pooling ratos for SAGPool and EdgePool.

MUTAG

The optimal configurations for MUTAG favored moderate pooling and clustering. `DMoN` performed best with $k = 16$, and `ECPool` retained 80% of the nodes. `TopK` used a 0.3 retention ratio, while `SAGPool` kept 90% of nodes. `Set2Set` operated with 3 steps, and `DiffPool` used $k = 19$. These settings reflect the small graph sizes and binary classification task, with minimal node loss and simple pooling.

PROTEINS

For the more complex PROTEINS dataset, the best results came from deeper aggregation and moderate pooling. `DMoN` increased to $k = 18$, `ECPool` reduced to 0.6, and `Set2Set` used 16 processing steps to capture richer structure. `TopK` retained 40%, `SAGPool` remained at 90%, and `DiffPool` stayed at $k = 19$. This setup balanced expressiveness and regularization.

ENZYMES

ENZYMES required aggressive pooling and lower cluster counts due to its complexity and multi-class nature. `DMoN` used a reduced $k = 11$, `ECPool` pooled only 10% of nodes, and `Set2Set` reverted to 3 steps. `TopK` retained 40%, `SAGPool` used a 0.8 ratio, and `DiffPool` also clustered into $k = 11$. These settings helped manage the graph complexity while avoiding overfitting.

We obtained maximum accuracy for all datasets when the pooling ratio was relatively high at 0.8-0.9.

4 Discussion

We find that DiffPool achieves highest accuracy but at increased time and memory cost. SAGPool offers a good trade-off. Under perturbations, DiffPool and SAGPool are most robust. The grid search indicates moderate pooling ratios (0.25–0.5) perform best.

- **Accuracy vs. Efficiency:** Differentiable pooling (DiffPool) and set2set yield high test accuracies (86.8% on MUTAG, 74.0% on PROTEINS, 44.2% on ENZYMES) but incurs substantial overhead in training time and GPU memory usage. In contrast, simpler methods such as global-sum and global-mean achieve competitive performance on MUTAG and ENZYMES while requiring only a fraction of the resources, making them attractive for smaller graphs or limited hardware environments.
- **Balanced Trade-offs:** SAGPool and DMoN strike an effective balance, offering near–state-of-the-art accuracy (78.9%–84.2% on MUTAG and PROTEINS) with moderate time and memory costs. Their attention- and community-based strategies dynamically select informative substructures, explaining their strong generalization across tasks.
- **Robustness to Perturbations:** Under structural (edge/node dropout) and feature noise perturbations on the ENZYMES dataset, DiffPool and set2set exhibit relatively higher degradation in accuracy.

These observations underscore that no single pooling mechanism universally dominates; rather, the choice should be guided by the target graph scale, complexity, and available computational budget.

5 Conclusion

Our core hypothesis—that different graph-level pooling mechanisms would yield varying trade-offs between accuracy and efficiency—was validated. Specifically:
High-capacity methods (DiffPool, Set2Set) achieved the best classification accuracy across all datasets. **Lightweight global pooling** (mean, max, sum) delivered reasonably strong performance (within 3–5% of the best) while requiring only a fraction of the compute and memory.
Intermediate approaches (SAGPool, TopKPool) balanced accuracy and cost, outperforming simple pooling on larger graphs and closing much of the gap to DiffPool.

Our ablation experiments—varying pooling ratios and introducing perturbations—yielded clear, actionable insights:
Optimal pooling ratios for simple/global methods lie in the 0.25–0.5 range, preserving sufficient graph information while reducing complexity. **DiffPool and SAGPool** tolerate much higher pooling ratios (0.8–0.9) before performance degrades, confirming their robustness. **Robustness tests** (node/edge removals, feature noise) showed that high-capacity pooling methods maintain >90% of their original accuracy under moderate perturbations, whereas global pooling methods degrade more sharply.

References

[1] Filippo Maria Bianchi and Veronica Lachi. The expressive power of pooling in graph neural networks. *Advances in neural information processing systems*, 36:71603–71618, 2023.

[2] Daniele Grattarola, Daniele Zambon, Filippo Maria Bianchi, and Cesare Alippi. Understanding pooling in graph neural networks. *IEEE transactions on neural networks and learning systems*, 35(2):2708–2718, 2022.

[3] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *International conference on machine learning*, pages 3734–3743. pmlr, 2019.

[4] Diego Mesquita, Amauri H. Souza, and Samuel Kaski. Rethinking pooling in graph neural networks. In *Advances in Neural Information Processing Systems*, volume 33, pages 2220–2231. Curran Associates, Inc., 2020.

A Pooling Mechanisms

A brief description of the pooling mechanisms we implemented and evaluated:

- **no-pool**: No pooling and node features $\{\mathbf{h}_v\}_{v\in V}$ are fed directly to the classifier.
- **global-max**: Takes the element-wise maximum over all node embeddings. This highlights the strongest activation across nodes for each feature dimension, capturing the most prominent local pattern but discarding subtler signals

$$[\mathbf{h}_G]_i = \max_{v\in V} [\mathbf{h}_v]_i.$$

- **dmon** (GNNDMoN): Adds a modularity-based loss to encourage community structure:

$$S = \mathrm{softmax}(\mathrm{GNN}_{\mathrm{assign}}(X, A)), \quad X' = S^\top X, \quad A' = S^\top A\, S,$$

  with modularity loss

$$\mathcal{L}_{\mathrm{mod}} = -\frac{\mathrm{tr}(S^\top A\, S)}{2m}.$$

  The learned S both pools and aligns with classical network modularity, yielding clusters that reflect community divisions

- **ecpool** (EdgePooling): Scores edges and contracts the top ρ—E— edges: merging their endpoints into supernodes and updating adjacency. By pooling on edges rather than nodes, it preserves connectivity and naturally reduces both nodes and edges, but deciding merges can be complex.

$$s_{uv} = \mathbf{w}^\top [\mathbf{h}_u \parallel \mathbf{h}_v],$$

- **mincut** (GNNMinCut): Learns a soft assignment S and then optimizes a "min-cut" objective:

$$S = \mathrm{softmax}(\mathrm{GNN}_{\mathrm{assign}}(X, A)), \quad \mathcal{L}_{\mathrm{mincut}} = -\frac{\mathrm{tr}(S^\top A\, S)}{\mathrm{tr}(S^\top D\, S)},$$

$$X' = S^\top X, \quad A' = S^\top A\, S.$$

  This encourages clusters that are both internally dense and well separated, yielding interpretable coarsenings at the cost of a complex loss term.

- **global-att** (GNNGlobalAttention): Learns attention weights over nodes, then computes a weighted sum:

$$\alpha_v = \mathrm{softmax}(\mathbf{a}^\top \tanh(W\mathbf{h}_v)), \quad \mathbf{h}_G = \sum_{v\in V} \alpha_v\, \mathbf{h}_v.$$

  Learns which nodes matter most for the task and can focus on relevant substructures at the expense of additional parameters and computation.

- **set2set** (GNNSet2Set): A sequence-to-sequence readout that alternates LSTM "queries" and attention over node features for T steps

$$q_t = \mathrm{LSTM}(q_{t-1}, r_{t-1}), \quad r_t = \sum_{v \in V} \mathrm{softmax}(\mathbf{h}_v^\top q_t)\, \mathbf{h}_v,$$

then

$$\mathbf{h}_G = [\, r_T ; q_T \,].$$

The final graph embedding captures complex, multi-step interactions but is more costly and introduces recurrence.

- **global-mean**:

$$\mathbf{h}_G = \frac{1}{|V|} \sum_{v \in V} \mathbf{h}_v.$$

This normalizes for graph size and treats every node equally, producing a smooth summary that can undervalue rare but important nodes.

- **global-sum**:

$$\mathbf{h}_G = \sum_{v \in V} \mathbf{h}_v.$$

Unlike mean, this preserves information about graph size but can be dominated by large graphs or high-magnitude features

- **topk** (GNNTopK): Scores each node via a learned projection and keeps only the top k nodes:

$$y_v = \mathbf{p}^\top \mathbf{h}_v, \quad \mathcal{I} = \mathrm{topk}(y, k), \quad X' = X[\mathcal{I}], \quad A' = A[\mathcal{I}, \mathcal{I}].$$

- **sag** (GNNSAG): Combines graph convolution and attention to score nodes, then applies Top-K. Surviving nodes' embeddings are re-weighted by their attention scores. This contextual scoring is more robust than simple projections.

$$y = \mathrm{GNN}_{\mathrm{att}}(X, A), \quad \alpha = \sigma(y), \quad \mathcal{I} = \mathrm{topk}(\alpha, \rho|V|),$$

$$X' = X[\mathcal{I}] \odot \alpha[\mathcal{I}], \quad A' = A[\mathcal{I}, \mathcal{I}].$$

- **diff** (GNNDiffPool): Differentiable pooling via a learned soft assignment matrix. With auxiliary entropy or link-prediction regularization, it provides fully end-to-end hierarchical learning, but can be memory-intensive for large graphs.

$$S = \mathrm{softmax}(\mathrm{GNN}_{\mathrm{assign}}(X, A)), \quad X' = S^\top X, \quad A' = S^\top A\, S,$$