Hierarchical Graph Representation Learning with Pooling-Induced Substructures

Luca Sbicego[*]
EPFL
luca.sbicego@epfl.ch

Xiaowen Dong
University of Oxford
xdong@robots.ox.ac.uk

Dorina Thanou
EPFL
dorina.thanou@epfl.ch

Abstract

Hierarchical graph-level tasks that require reasoning over substructures that extend beyond local neighborhoods are posing challenges for standard Graph Neural Networks (GNNs). While graph pooling aims to address this by coarsening graphs into hierarchical representations, existing methods often fail to explicitly preserve the substructures they induce, leading to inconsistent empirical performance. Moreover, evaluating such approaches remains challenging due to the lack of benchmarks with explicit hierarchical structure. We propose Pooled-Substructure-Aware GNNs, an approach to graph pooling that explicitly preserves and processes substructures uncovered by structure-based pooling schemes. We show that this design enhances the expressive power of pooling-based GNNs in the sense of the Weisfeiler-Lehman (WL) test. To enable principled evaluation, we introduce two new synthetic datasets with controllable hierarchical structure and propose a metric to characterize the prevalence of hierarchical patterns in real-world graph-level benchmarks. On tasks with relevant hierarchical structure, our approach outperforms matched pooling baselines, while remaining competitive on structurally simpler benchmarks. Code and datasets are publicly available at https://github.com/lsbicego/PSA-GNN/.

1 Introduction

Graph Neural Networks (GNNs) are widely used to process graph-structured data [1, 2], but their prevalent message passing formulation introduces a strong locality bias [3, 4]. Many graph tasks, however, require reasoning over structures extending beyond local neighborhoods, such as communities, or other forms of hierarchical organization [5–7] (Figure 1, left panel). While Message Passing Neural Networks (MPNNs) and Graph Transformers (GTs) can capture such patterns in principle, they must infer the underlying hierarchical organization implicitly, which can be challenging. Graph pooling has emerged as a standard mechanism for hierarchical graph representation learning [8], where graph convolutions are interleaved with pooling layers, which coarsen the input graph by grouping nodes into supernodes based on connectivity or learned node features [9].

In principle, this should enable multi-scale reasoning, making graph pooling a natural foundation for hierarchical modeling. However, in practice, pooling often yields limited or inconsistent gains over simpler flat MPNNs [10]. Widely used schemes that learn to coarsen a graph based on its node features in an end-to-end fashion have been shown to perform on par with randomized implementations that assign nodes to supernodes by chance [10]. Similarly, this type of pooling does not increase the expressivity of the underlying graph convolution in terms of the Weisfeiler-Lehman (WL) test, but at best preserves it [11, 12]. In contrast, structure-based pooling schemes that group nodes into hard clusters based on connectivity patterns inaccessible to standard MPNNs have been shown to increase expressivity [11]. This suggests that graph pooling can expose meaningful higher-order structure

[*]Work conducted while at the University of Oxford.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

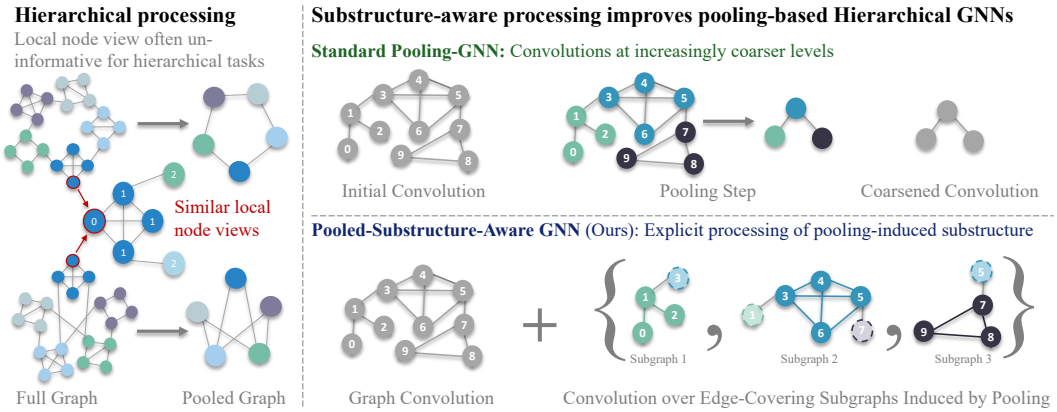


Figure 1: **Motivation and overview.** Hierarchical tasks are challenging to solve for node-based MPNNs and GTs. We introduce PSA-GNN, a framework to process meaningful substructures exposed by graph pooling that is more expressive and improves performance on hierarchical graph-level tasks.

beyond simple coarsening of the input graph. Concretely, each supernode can be viewed as exposing a substructure, given by the induced subgraph over its constituent nodes and their internal edges. Yet, existing pooling-based approaches ignore this exposed structure: they collapse each cluster into a single supernode and discard the corresponding subgraph (Figure 1, top right panel).

At the same time, evaluating graph pooling approaches remains a significant challenge. Hierarchical GNNs are typically assessed on standard benchmarks, where the extent of hierarchical patterns relevant to the graph-level label is unclear [13, 14]. Synthetic datasets with controllable hierarchical structure and difficulty are missing, making it difficult to isolate when and why pooling is beneficial.

These observations reveal both an architectural and an empirical gap in hierarchical graph learning: pooling-induced substructures are not explicitly preserved, and existing benchmarks make it difficult to assess when such structure matters. We address these gaps through the following contributions.

- **Novel Framework for Hierarchical Modeling Using Graph Pooling (Section 3.1).** We introduce Pooled-Substructure-Aware GNN (PSA-GNN; Figure 1, bottom right panel), a principled framework that processes pooling-induced clusters as explicit substructures rather than collapsing them into supernodes, enabling richer hierarchical representations.
- **Theoretical Expressivity Analysis (Section 3.2).** We provide a theoretical analysis showing that incorporating pooling-induced substructures increases expressivity in the WL sense compared to standard pooling.
- **Hierarchical Benchmarks (Section 4.1).** We introduce two synthetic datasets, MACRO-HIERARCHIES and GRANULAR-HIERARCHIES, enabling a controlled evaluation of hierarchical reasoning capabilities.
- **Principled Evaluation.** We further present a first characterization of the presence of hierarchical substructures in real-world benchmarks based on Nested Stochastic Block Models (Section 4.2). Together with our hierarchical benchmarks, this leads to a clear interpretability of empirical performances: on synthetic benchmarks and real-world datasets exhibiting hierarchical structure, our approach improves over existing pooling baselines, while remaining competitive on structurally simpler benchmarks (Section 5).

2 Background

Notation. Let an undirected, unweighted graph be denoted by $G = (V, E)$, where V and E are the node and edge sets, respectively, and $|V| = N$. We define the adjacency matrix $\mathbf{A} \in \{0, 1\}^{N \times N}$ and node features $\mathbf{X} \in \mathbb{R}^{N \times F}$, where F is the feature dimension. When comparing the expressivity of models, we use the following definition.

Definition 1 (Expressivity Comparisons, following [11])**.** Let θ, ϕ be two permutation-invariant functions mapping graphs to vectors. Let G, H be any two graphs. Then:

- θ is *at least as expressive as* ϕ if $\phi(G) \neq \phi(H) \implies \theta(G) \neq \theta(H)$.
- θ and ϕ are *equally expressive* if $\phi(G) \neq \phi(H) \iff \theta(G) \neq \theta(H)$.
- θ is *more expressive than* ϕ if θ is at least as expressive as ϕ and there exists a pair of graphs G, H such that $\theta(G) \neq \theta(H)$ but $\phi(G) = \phi(H)$.

Graph pooling. A graph pooling operator maps a graph G to a coarsened graph G' with $|V'| < |V|$. We adopt the standard SEL-RED-CON view of pooling [9]. Nodes are first assigned to supernodes via SEL, node features are then aggregated within each supernode by RED, and connectivity between supernodes is determined by CON. Hierarchical GNNs interleave such pooling steps with message passing in order to process a graph at increasingly coarser levels.

Graph pooling in theory and practice. Graph pooling has often delivered inconsistent empirical gains, with hierarchical architectures frequently failing to outperform simpler flat baselines [10]. Many popular pooling operators are node-based, with assignments or scores learned from the same local node representations produced by the underlying MPNN. Recent expressivity results show that such pooling schemes cannot exceed the expressive power of the base encoder when their assignments depend only on these node representations [11, 12]. This may help explain why learned pooling can fail to introduce new structural information. These observations motivate us to investigate pooling from both theoretical and benchmarking perspectives.

Structure-based pooling. By contrast, recent work has shown that pooling schemes determining cluster assignments based on graph structure rather than learned node features *can* increase expressivity [11]. In this setting, pooling-induced clusters can be viewed not only as supernodes for coarsening, but also as structurally meaningful subgraphs of the original graph. This perspective forms the basis of our approach.

Subgraph GNNs. Subgraph GNN methods [15–18] aim to improve graph representations by processing a graph through a collection of overlapping subgraphs rather than only through the full graph itself. Intuitively, this allows nodes to be represented in multiple local contexts, strengthening expressivity beyond standard message passing [15, 16]. This raises the question of how to exploit pooling-induced subgraphs.

3 From Graph Pooling to Substructure-Aware Processing

We introduce a principled framework for processing *meaningful pooling-induced substructures*, instantiated by Pooled-Substructure-Aware GNNs (PSA-GNN). The key idea is to treat each pooled cluster as an induced subgraph rather than immediately collapsing it into a supernode. This contrasts with standard graph pooling, which reduces each cluster to a supernode and continues computation on the coarsened graph, thereby discarding the internal structure induced by the clusters.

3.1 Processing Pooling-Induced Substructures

Our perspective naturally connects graph pooling to the Subgraph GNN setting, where models operate not only on the input graph itself, but also on a collection of subgraphs extracted from it [15–18]. In our setting, the subgraph collection is not generated by a separate subgraph selection rule, but arises directly from the substructures exposed by the pooling operator:

Definition 2 (Pooling-induced subgraph policies)**.** Let a graph pooling scheme partition the nodes of $G = (V, E)$ into disjoint clusters $\mathcal{A}_1, \ldots, \mathcal{A}_m$. A subgraph policy π induces a set of subgraphs $\mathcal{S} = \{G_i = (V_i, E_i)\}_{i=1}^m$ with $V_i = \mathcal{A}_i$ and $E_i = \{(u, v) \in E : u, v \in V_i\}$. Note that the induced subgraphs are vertex-disjoint. We further consider a padded policy $\hat{\pi}$ that augments each subgraph with crossing edges and their incident boundary nodes, yielding an edge-covering set of subgraphs.

Building on these pooling-induced subgraph policies, we introduce PSA-GNN, which processes the pooling-induced subgraphs created by a policy π using a subgraph-processing layer inspired by DSS-GNN [15]. The layer combines subgraph-level updates with a graph-level context update, as illustrated in Figure 2. For each subgraph, a shared encoder L^{sg} updates node representations using only the internal structure of that subgraph. In parallel, subgraph-node features are scattered back to the original node space, processed on the original graph by L^{graph}, and then mapped back to each subgraph. This graph-level branch allows information to flow across different pooled subgraphs through the original graph adjacency.

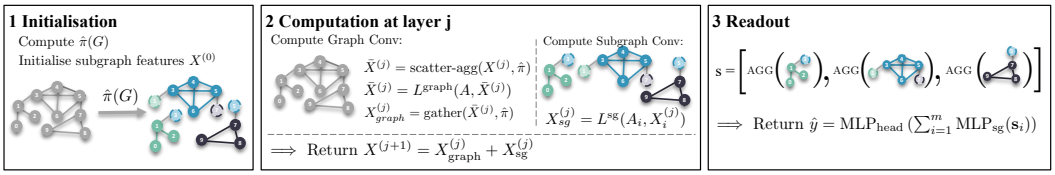


Figure 2: **Computational flow of PSA-GNN.** We pre-compute the subgraph policy and initialize node features in subgraph space. Each layer then updates these representations by combining message passing on the original graph with message passing inside each induced subgraph. Finally, node representations are aggregated within each subgraph and passed to a DeepSets-style readout.

Formally, for each subgraph $i \in \{1, \ldots, m\}$, layer L updates the node feature matrix as

$$\mathbf{X}'_i = L(\mathbf{A}, \mathbf{X}, \pi)_i = L^{\text{sg}}(\mathbf{A}_i, \mathbf{X}_i) + \left(L^{\text{graph}}(\mathbf{A}, \bar{\mathbf{X}})\right)_{\pi_i}, \tag{1}$$

where $\mathbf{A}_i$ and $\mathbf{X}_i$ denote the adjacency and node feature matrices of subgraph i, and $\mathbf{A}$ is the adjacency matrix of the original graph. The matrix $\bar{\mathbf{X}}$ is obtained by scatter-aggregating all subgraph-node features back to the original node space. For vertex-disjoint subgraphs, this is simply a re-indexing step; for overlapping subgraphs, duplicate occurrences are averaged. The operator $(\cdot)_{\pi_i}$ denotes gathering original-node features to the nodes in subgraph i.

For graph-level prediction, we first pool the node features of each subgraph into a subgraph representation $\mathbf{s}_i$, and then apply a DeepSets-style readout [19]:

$$\hat{y} = \text{MLP}_{\text{head}}\left(\sum_{i=1}^{m} \text{MLP}_{\text{sg}}(\mathbf{s}_i)\right). \tag{2}$$

Alternatively, the element-wise transformation MLP_{sg} can be replaced by a GNN operating over the pooled graph: $\text{GNN}_{\text{sg}}(\mathbf{A}', \mathbf{Z})$, where $\mathbf{Z}_i = \mathbf{s}_i$ and $\mathbf{A}'$ is the adjacency matrix of the coarsened graph.

Unless otherwise specified, PSA-GNN uses a DeepSets-style readout and the padded variant from Definition 2, which augments each subgraph with crossing edges and their associated boundary nodes. This introduces controlled overlap between neighboring pooled substructures, allowing local subgraph updates to access boundary context while the graph-level branch couples information through the original graph adjacency. To distinguish original cluster nodes from added boundary nodes, we append a binary node feature indicating whether a node belongs to the core cluster or was introduced through padding.

3.2 Expressivity of Explicit Substructure Processing

In this section, we establish an expressivity hierarchy that characterizes the benefits of explicitly processing pooling-induced substructures. In particular, we investigate how within-subgraph message passing and information exchange across substructures contribute to the expressive power of pooling-based GNNs.

To this end, we consider two comparison architectures. First, we introduce IP-GNN (Intra-Pooling-GNN), a simpler variant that uses the base policy π to produce disjoint clusters and extends the standard SEL-RED-CON pooling pipeline by performing message passing within each pooling-induced subgraph before the RED and CON steps. Second, we consider a standard SEL-RED-CON Pool-GNN as a matched pooling baseline.

We first analyze the effect of within-subgraph message passing. Proposition 3 formalizes the basic intuition that immediate coarsening can discard information already present in the substructures induced by pooling. Even a single message-passing step restricted to these subgraphs can therefore strengthen pooling.

Proposition 3. Let $\mathcal{C} = \mathcal{A}_1, \ldots, \mathcal{A}_m$ be a fixed partition of the node set into disjoint clusters. Then, IP-GNN is at least as expressive as Pool-GNN in the WL sense. Moreover, there exists a partition $\widetilde{\mathcal{C}}$ for which IP-GNN is strictly more expressive than Pool-GNN.

We next show that PSA-GNN subsumes the simpler within-subgraph construction and can go strictly beyond it once pooled subgraphs are allowed to overlap through padding. Intuitively, this overlap

enables the model to couple information across neighboring pooled substructures while preserving the underlying cluster structure identified by pooling.

Proposition 4. For any subgraph policy π, PSA-GNN with supergraph readout is at least as expressive as IP-GNN under the same policy. Moreover, there exists a policy π such that PSA-GNN using the corresponding padded, edge-covering policy $\hat{\pi}$ together with appropriate node markings and any injective readout is strictly more expressive than IP-GNN under π.

Propositions 3 and 4 in combination yield the hierarchy between PSA-GNN, IP-GNN, and Pool-GNN. Full formal statements, proofs, and additional examples are deferred to Appendix A. Finally, we verify this relationship empirically on BREC, a standard expressivity benchmark in Appendix B.

4 Characterizing Hierarchical Graph Structure

The results of Section 3 suggest that explicit substructure processing can increase expressivity by leveraging graph pooling more effectively. We expect these gains to translate into improved downstream performance particularly when graph-level tasks depend on hierarchically organized substructures in the input graph rather than local node-level patterns. Figure 1 (left panel) illustrates this in an example: two graphs may exhibit identical local neighborhoods around selected nodes while differing in their global community organization. In such situations, local node-level views computed by bounded-depth MPNNs may be uninformative. Instead, GTs can enrich node-level representations with positional or structural encodings, but these mechanisms are not designed to explicitly identify task-relevant structures. Thus, while both model families can potentially capture higher-order patterns, they must learn to identify the relevant structures implicitly, whereas substructure-aware models can process them directly.

However, graph pooling methods are typically evaluated on standard graph-level benchmarks, where the relevance of hierarchical structure to the prediction task is often unclear [10, 20, 13]. To address this limitation, we adopt two complementary evaluation strategies: controlled synthetic benchmarks with explicit hierarchical signals, and a structural characterization of real-world datasets.

4.1 Synthetic Hierarchical Benchmarks

To study hierarchical graph modeling in a controlled setting, we introduce two synthetic benchmarks in which graph labels explicitly depend on hierarchical structure. The datasets are generated using SBMs [21]; full construction details are deferred to Appendix D.1. By design, the resulting prediction tasks do not impose an expressivity barrier and remain solvable, in principle, by sufficiently deep MPNNs.

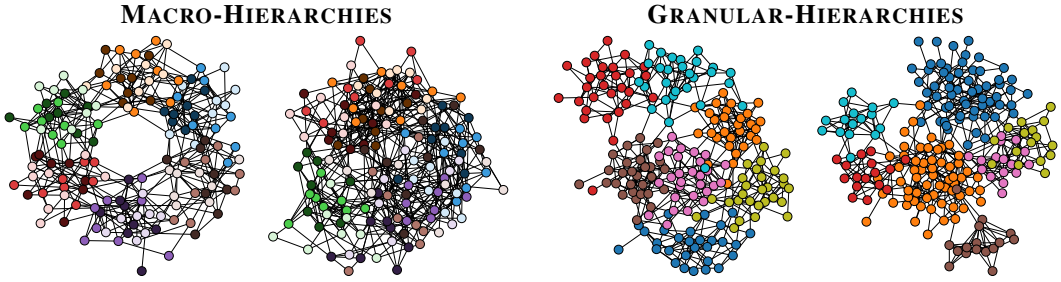
MACRO-HIERARCHIES tests whether a model can reason about higher-level connectivity between clusters, inspired by the example in Figure 1. Each graph is generated from a two-level SBM, with micro-communities grouped into macro-communities. The class signal is encoded at the macro level: macro-communities are arranged either in a ring or in a bipartite pattern, as illustrated in Figure 3. Distinguishing the two classes therefore requires distinguishing different interaction patterns between larger substructures. We control task difficulty by adding random inter-cluster edges that obscure the underlying ring or bipartite pattern.

GRANULAR-HIERARCHIES complements the above dataset and tests whether a model can reason about the size distribution within clusters themselves. Graphs with class $y = 0$ consist of equally sized clusters, whereas graphs with class $y = 1$ contain a mixture of small and large clusters, as illustrated in Figure 3. The signal therefore lies in the relative granularity of the substructures. As in MACRO-HIERARCHIES, we can control the difficulty of this task by adding more inter-cluster edges and thus making the substructure signal noisier.

4.2 Structural Characterization of Real-World Benchmarks

Complementing our controlled synthetic benchmarks, we characterize the extent of hierarchical structure in real-world graph-level datasets. This provides a structural perspective for interpreting when explicit substructure processing may be beneficial in practice.

For each graph, we fit a degree-corrected nested SBM to its undirected, unweighted topology [21–23], from which we extract two descriptors: the detected *substructure count*, i.e., the number of



(a) Ring (b) Bipartite (c) Even Clusters (d) Uneven Clusters

Figure 3: **Examples from two synthetic datasets.** Left: MACRO-HIERARCHIES, where the class distinction is given by ring versus bipartite higher-level structure. Right: GRANULAR-HIERARCHIES, where the class distinction is given by even versus uneven cluster sizes.

blocks at the finest inferred level, and the *substructure depth*, i.e., the number of inferred hierarchy levels. Together, these descriptors quantify how much partitionable substructure and hierarchical organization the graph exhibits under the fitted model.

Figure 4 shows the distribution of these descriptors across graphs from several widely used graph-level benchmarks. Under this structural probe, ZINC [24] and NCI1 [25] appear structurally shallow, as almost all graphs are assigned a single detected substructure at depth 1. In contrast, REDDIT-5K, REDDIT-12K, COLLAB [25], and MalNet-Tiny [26] exhibit substantially richer inferred group structure and, for many graphs, multiple hierarchy levels. We use these findings to distinguish structurally flatter datasets from more structure-dominated benchmarks in the experiments below. To validate the methodology and to provide a point of reference, we also include an instance of our synthetic datasets at a medium difficulty level. As expected, the fitted models recover clear hierarchical structure in these graphs.

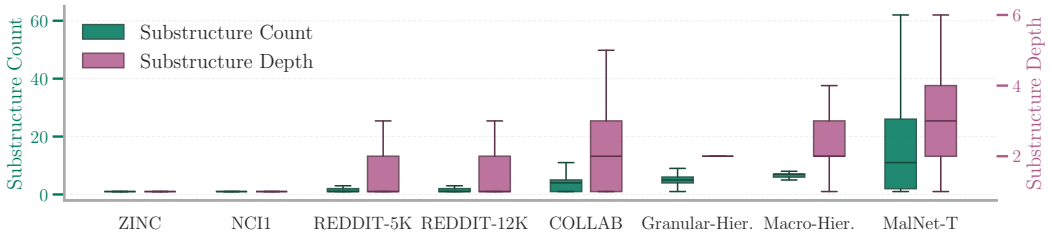


Figure 4: **Structural characterization of real-world benchmarks using nested SBM fits.** For each graph, we fit a degree-corrected nested SBM and record the number of detected substructures at the finest inferred level (left axis) and the detected substructure depth (right axis).

5 Experiments

We evaluate whether the expressivity gains from explicit substructure processing translate into improved graph-level performance when hierarchical structure is relevant to the prediction task. In all experiments, we compare PSA-GNN to the simple within-subgraph baseline (IP-GNN) as well as the standard pooling baseline (Pool-GNN) which immediately coarsens the graph, using matched pooling policies.

Setup. Our comparisons are centered on three structure-based pooling schemes: Leiden [27], a community-detection method; CurvPool [28], which clusters nodes using balanced Forman curvature; and CliquePool [29], which forms clusters from maximal cliques. For each policy, we evaluate the three pooling-based architectures for a single pooling step under matched depth and parameter budgets. Throughout all experiments, we use GIN [4] as the 1-WL base message passing layer. Hyperparameters are chosen on validation data, and the same dataset-specific training protocol is used across models. Full details and implementation choices are provided in Appendix E. Additional results for exploratory pooling-based subgraph policies are provided in Appendix F.

5.1 Controlled Hierarchical Benchmarks

The synthetic benchmarks introduced in Section 4.1 provide a controlled setting for testing our hypothesis, as graph labels explicitly depend on hierarchical graph structure. In addition to the matched pooling variants Pool-GNN, IP-GNN, and PSA-GNN, we compare against a flat MPNN based on GIN [4] and a GT baseline based on GPS [30]. Figure 5 reports test accuracy across Leiden, CurvPool, and CliquePool across varying difficulty levels.

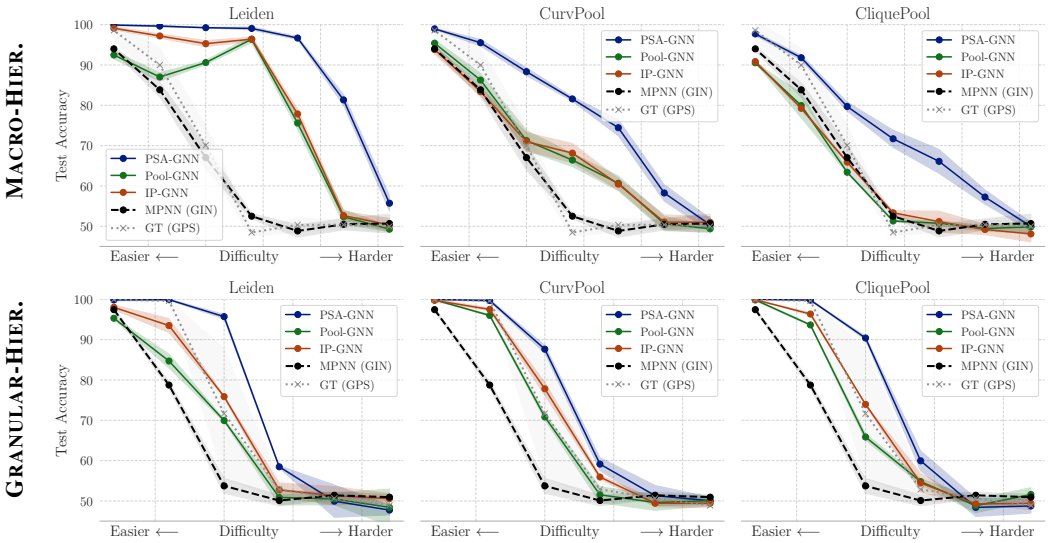


Figure 5: **Results on the synthetic hierarchical benchmarks.** We compare Pool-GNN, IP-GNN, and PSA-GNN against flat MPNN and GT baselines across representative structure-based schemes.

Across both synthetic benchmarks, PSA-GNN achieves the best performance across the evaluated difficulty levels and remains substantially more robust as the hierarchical signal becomes harder to recover. The flat MPNN and GPS baselines often degrade more rapidly as difficulty increases, supporting the motivation that node-level models can struggle to infer the relevant higher-order organization implicitly.

The matched pooling variants further show the value of preserving the structures identified by pooling for tasks that depend on higher-order graph organization: IP-GNN often improves over immediate coarsening, while PSA-GNN achieves the strongest performance by processing pooling-induced substructures end-to-end. This pattern is consistent with the expressivity hierarchy established in Section 3.2 and provides controlled evidence that explicit substructure processing is useful when the task depends on hierarchical graph organization. The magnitude of these gains varies across pooling policies, highlighting that the benefits also depend on how well the resulting substructures align with the task-relevant hierarchy.

We additionally evaluate common learned pooling operators on the synthetic benchmarks in Appendix C.1. Sparse methods such as SAGPool and ASAPool [31, 32] can perform strongly on MACRO-HIERARCHIES, while dense methods like MinCutPool or DiffPool [33, 34] do not improve over baselines. These results highlight the diagnostic value of the benchmarks and suggest that different pooling mechanisms capture different hierarchical signals. We further assess the impact of informative node features by repeating the medium-difficulty experiments with contextual SBM features [35] in Appendix C.2. Informative features narrow the performance gap when directly relevant to the task, while PSA-GNN retains its advantage on higher-level structural reasoning.

5.2 Real-World Benchmarks

Table 1 presents results on the real-world benchmarks characterized in Section 4.2. We again focus on the matched comparison between PSA-GNN, IP-GNN, and Pool-GNN under the same three structure-based pooling policies. We additionally include popular hierarchical GNN baselines, namely ASAPool, SAGPool, MinCutPool, and DiffPool, as well as a flat MPNN baseline. Where

computationally feasible, we also evaluate a Subgraph GNN baseline that applies PSA-GNN to all 2-hop egonets. This provides a reference for subgraph processing with a generic, non-pooling-based subgraph policy, while the matched pooling variants remain the focus of our architectural comparison.

Table 1: **Results on real-world benchmarks.** Best overall result per dataset is highlighted in **brown**, second best in **purple**. Among PSA-GNN, IP-GNN, and Pool-GNN, the best method for a given subgraph policy π is shown **bolded**. A $\star$ indicates that PSA-GNN significantly outperforms the matched Pool-GNN under a two-sided Welch test ($p < 0.05$, unadjusted). The *Avg. Rank* reports the average rank of each architecture level across matched dataset–policy comparisons among PSA-GNN, IP-GNN, and Pool-GNN. Dense pooling schemes do not scale and ASAPool cannot use edge features by default.

			Hierarchical Structure				Flat Structure	
Model	**Subgraph Policy**	**Avg. Rank (↓)**	**COLLAB** *Acc. [%] (↑)*	**REDDIT-5K** *Acc. [%] (↑)*	**REDDIT-12K** *Acc. [%] (↑)*	**MalNet** *Acc. [%] (↑)*	**NCI1** *Acc. [%] (↑)*	**ZINC** *MAE (↓)*
PSA-GNN	CliquePool		**78.1 ± 1.3**$\star$	**55.0 ± 0.8**	47.8 ± 1.4	90.6 ± 2.4	**81.5 ± 2.5**	**0.182 ± 0.008**
	CurvPool	**1.36**	76.8 ± 1.5$\star$	**54.9 ± 1.7**	**50.1 ± 1.7**$\star$	**92.0 ± 1.9**	81.2 ± 1.6	**0.177 ± 0.005**
	Leiden		**76.4 ± 2.2**	**55.3 ± 2.1**$\star$	**50.0 ± 1.3**$\star$	**91.1 ± 1.4**$\star$	80.0 ± 2.5	0.213 ± 0.007$\star$
IP-GNN	CliquePool		75.0 ± 2.3	53.3 ± 3.4	45.1 ± 2.1	**91.7 ± 1.7**	80.6 ± 2.1	0.183 ± 0.004
	CurvPool	2.33	**77.4 ± 0.9**	52.6 ± 3.3	42.0 ± 2.5	89.3 ± 2.0	80.3 ± 1.9	0.184 ± 0.005
	Leiden		73.6 ± 3.1	52.2 ± 2.6	45.6 ± 0.8	90.7 ± 1.1	**80.1 ± 2.6**	**0.186 ± 0.004**
Pool-GNN	CliquePool		70.4 ± 4.1	52.4 ± 2.9	**48.3 ± 1.7**	90.2 ± 1.6	81.0 ± 2.4	**0.182 ± 0.004**
	CurvPool	2.31	71.1 ± 4.5	53.6 ± 2.6	47.4 ± 0.4	91.3 ± 1.1	**81.3 ± 2.0**	0.178 ± 0.004
	Leiden		74.1 ± 2.3	53.1 ± 1.9	44.7 ± 1.6	87.9 ± 1.5	79.9 ± 2.0	0.257 ± 0.013
Hierarchical GNN	ASAPool	–	71.6 ± 5.4	–	49.5 ± 0.2	88.2 ± 3.1	81.2 ± 2.0	–
	SAGPool	–	70.6 ± 4.2	–	46.6 ± 1.8	90.1 ± 1.1	80.1 ± 1.9	0.416 ± 0.107
	MinCutPool	–	–	–	–	–	80.2 ± 2.0	0.161 ± 0.003
	DiffPool	–	–	–	–	–	81.0 ± 1.7	0.147 ± 0.005
Baselines	Egonet	–	–	–	–	91.4 ± 1.4	81.6 ± 2.4	0.403 ± 0.060†
	MPNN (GIN)	–	74.9 ± 3.7	53.4 ± 2.4	48.3 ± 2.3	91.0 ± 1.6	80.9 ± 2.5	0.172 ± 0.005

† The Egonet result on ZINC is likely under-converged under the shared training budget.

Across these real-world benchmarks, explicit substructure processing strengthens structure-based pooling. Aggregated over all dataset–policy comparisons, PSA-GNN achieves the best mean rank, ahead of IP-GNN and Pool-GNN (1.36 vs. 2.33 and 2.31 respectively), suggesting that its benefits extend across different pooling policies. These results position PSA-GNN as an effective architecture for real-world hierarchical graph modeling.

The choice of subgraph policy nevertheless remains task-dependent, as different schemes expose different structural patterns. Since no policy consistently dominates across datasets, domain knowledge or validation performance can guide its selection.

5.3 Validation of Structural Hierarchy Characterization

Figure 6 relates the performance gains of PSA-GNN to the structural characterization introduced in Section 4.2. For each dataset, we report the relative gain of PSA-GNN over the flat MPNN baseline using the best-performing structure-based pooling policy. In line with our hypothesis, the group-level average gain is positive on benchmarks with richer inferred hierarchical structure, but slightly negative on structurally flatter datasets. On COLLAB, REDDIT-5K, REDDIT-12K, and MalNet-Tiny, which exhibit richer inferred substructure, PSA-GNN achieves the strongest overall performance, suggesting that explicit substructure processing is particularly beneficial in these settings. On NCI1 and ZINC, PSA-GNN remains competitive, especially within the family of structure-based pooling models, although the overall performance differences are smaller. In these structurally flatter settings, MPNN baselines and pooling schemes with dense soft assignments, such as DiffPool and MinCutPool, can also perform strongly. Overall, these results are consistent with the topology-driven hierarchical structure detected by our SBM probe and support the use of our metric for interpreting when pooling is likely to help.

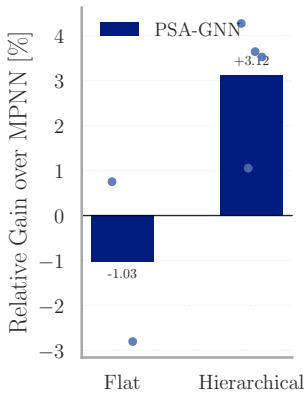


Figure 6: **Relative gain over MPNN.** Bars show group means; points show individual datasets. Positive values indicate improvement over MPNN.

Table 2: **Test MAE on GraphBench CO benchmarks.** We compare CO-ER (flat graphs) and CO-RB (with hierarchical structure) on 10K-graph subsets.

Model	Subgraph Policy	Flat CO-ER *MAE (↓)*	Hierarchical CO-RB *MAE (↓)*
PSA-GNN	CliquePool	0.5722 ± 0.0085	1.0108 ± 0.0145
	CurvPool	0.5917 ± 0.0177	0.9345 ± 0.0649
	Leiden	0.6015 ± 0.0145	1.2551 ± 0.0598
Baseline	MPNN (GIN)	0.5844 ± 0.0347	1.4251 ± 0.0180
Avg. Gain vs. MPNN		**-0.70%**	**+25.14%**

We further validate our approach on a combinatorial optimization (CO) task of predicting the objective value of a maximum independent set task [36]. CO-ER [37] consists of flat ER graphs (one substructure across one level), whereas CO-RB [38] contains hierarchical graphs according to our structural descriptors, with 22.7 substructures across 4 levels on average. This contrast clearly illustrates that our structural characterization is indicative of the benefits of explicit substructure processing, with PSA-GNN achieving larger performance gains on datasets exhibiting richer inferred hierarchical structure.

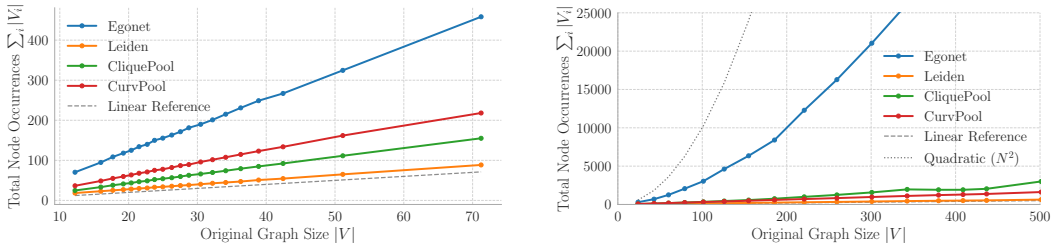
5.4 Scalability Analysis

Next, we analyze the scalability of PSA-GNN across different subgraph policies. A standard MPNN has time and memory complexity of $\mathcal{O}(N + M)$, where N and M denote the number of nodes and edges, respectively. Excluding subgraph construction, PSA-GNN additionally processes the pooling-induced substructures and thus incurs a policy-dependent overhead factor c_π:

$$\mathcal{O}\Big((1 + c_\pi)(|V| + |E|)\Big), \quad \text{with } c_\pi = \frac{\sum_i (|V_i| + |E_i|)}{N + M}. \tag{3}$$

The value of c_π depends on the size and overlap of the selected substructures. For example, for a disjoint partition π (see Def. 2), we have $\sum_i |V_i| = N$ and $\sum_i |E_i| \leq M$, so the subgraph branch adds at most one additional graph-sized computation.

Figure 7 displays the total number of subgraph-node occurrences $\sum_i |V_i|$ induced by different padded policies $\hat{\pi}$ as a function of the original graph size. This quantity drives the factor c_π and serves as a simple proxy for memory usage during training and inference.



(a) Scalability on **NCI1**

(b) Scalability on **REDDIT**

Figure 7: **Scalability of subgraph policies.** We compare the total number of subgraph-node occurrences $\sum_i |V_i|$ induced by different policies. Observations are grouped into 20 bins and averaged for readability; very large REDDIT graphs are omitted from the plot.

Pooling-induced policies generate substantially fewer subgraph-node occurrences than generic egonet policies, often by orders of magnitude on larger graphs. Processing these substructures therefore remains scalable, making pooled-substructure processing a promising approach for hierarchical modeling on larger graphs. Per-epoch training times are reported in Appendix E.3.

6 Related Work

Graph pooling. Many widely used pooling architectures [8, 9] rely on learned dense assignments or sparse node-selection mechanisms, making learned pooling a central paradigm in hierarchical GNNs. Motivated by recent results showing that structure-based pooling can increase expressive power [11], we revisit simpler pooling policies derived from graph topology. While some pooling methods use local subgraph operations for scoring or compression [32, 39], we propose a different view of structure-based pooling by leveraging the exposed substructures as central objects for hierarchical modeling.

Higher-order Graph Modeling. Our work also connects graph pooling to higher-order graph learning approaches that operate on structures beyond individual nodes. In particular, our architecture is inspired by Subgraph GNNs [15–18], which improve graph representations by processing collections of subgraphs. Rather than relying on generic subgraph policies such as egonets or node-deleted subgraphs, we leverage the clusters exposed by graph pooling, directly linking substructure processing to hierarchical graph modeling. This also yields collections of substructures that remain scalable to larger graphs, as is often required in hierarchical modeling. The same distinction applies to graph patching methods such as Graph MLP-Mixer [40], which tokenize input graphs into node groups. Finally, our work is conceptually related to hypergraph learning [41–43], where hyperedges can be viewed as meaningful overlapping node groups, providing a useful perspective for future extensions of our framework.

7 Conclusion

We introduced a pooled-substructure view of graph pooling by treating induced clusters as valuable substructures for hierarchical tasks. This approach improves hierarchical graph modeling in practice and strengthens pooling from an expressivity perspective. To enable principled evaluation, we proposed two new synthetic datasets with tasks requiring hierarchical reasoning and characterized hierarchical structure in real-world benchmarks. On tasks with meaningful hierarchical structure, PSA-GNN consistently outperforms immediate coarsening, showing that the substructures exposed by pooling are useful to preserve and process rather than intermediate artifacts to discard.

At the same time, our experiments clarify the scope of our method. Pooling is beneficial only when graphs exhibit hierarchical structure, and the pooling scheme must expose this structure appropriately, which may vary across datasets. Moreover, our structural characterization should be viewed as a first step toward quantifying hierarchy, as it does not yet assess whether the detected structure is directly predictive of the graph label.

Overall, our results position structure-based pooling combined with explicit substructure processing as a promising direction for hierarchical graph representation learning. Rather than serving solely as a coarsening mechanism, graph pooling can be viewed as a way to expose relevant substructures for computation. This opens a broader design space for hierarchical GNNs and reveals useful connections to other higher-order graph learning frameworks, which can inform future hierarchical modeling architectures.

Acknowledgments and Disclosure of Funding

This work received no specific external funding and the authors declare no competing interests.

References

[1] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[2] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021.

[3] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pages 1263–1272, 2017.

[4] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

[5] Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.

[6] Erzsébet Ravasz and Albert-László Barabási. Hierarchical organization in complex networks. *Physical Review E*, 67(2):026112, 2003.

[7] Aaron Clauset, Cristopher Moore, and Mark EJ Newman. Hierarchical structure and the prediction of missing links in networks. *Nature*, 453(7191):98–101, 2008.

[8] Chuang Liu, Yibing Zhan, Jia Wu, Chang Li, Bo Du, Wenbin Hu, Tongliang Liu, and Dacheng Tao. Graph pooling for graph neural networks: Progress, challenges, and opportunities. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pages 6712–6722, 2023.

[9] Daniele Grattarola, Daniele Zambon, Filippo Maria Bianchi, and Cesare Alippi. Understanding Pooling in Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

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

[11] Veronica Lachi, Alice Moallemy-Oureh, Andreas Roth, and Pascal Welke. Expressive pooling for graph neural networks. *Transactions on Machine Learning Research*, 2025.

[12] Filippo Maria Bianchi and Veronica Lachi. The expressive power of pooling in graph neural networks. In *Advances in Neural Information Processing Systems*, 2023.

[13] Pengyun Wang, Junyu Luo, Yanxin Shen, Ming Zhang, Shaoen Qin, Siyu Heng, and Xiao Luo. A comprehensive graph pooling benchmark: Effectiveness, robustness and generalizability. *arXiv preprint arXiv:2406.09031*, 2024.

[14] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. In *International Conference on Learning Representations*, 2020.

[15] Beatrice Bevilacqua, Fabrizio Frasca, Derek Lim, Balasubramaniam Srinivasan, Chen Cai, Gopinath Balamurugan, Michael M. Bronstein, and Haggai Maron. Equivariant subgraph aggregation networks. In *International Conference on Learning Representations*, 2022.

[16] Lingxiao Zhao, Wei Jin, Leman Akoglu, and Neil Shah. From stars to subgraphs: Uplifting any GNN with local structure awareness. In *International Conference on Learning Representations*, 2022.

[17] Muhan Zhang and Pan Li. Nested graph neural networks. In *Advances in Neural Information Processing Systems*, volume 34, pages 15734–15747, 2021.

[18] Fabrizio Frasca, Beatrice Bevilacqua, Michael Bronstein, and Haggai Maron. Understanding and extending subgraph GNNs by rethinking their symmetries. In *Advances in Neural Information Processing Systems*, volume 35, pages 31376–31390, 2022.

[19] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[20] Luca Sbicego, Sevda Öğüt, Manuel Madeira, Yiming Qin, Dorina Thanou, and Pascal Frossard. On the role of structure in hierarchical graph neural networks. In *Proceedings on "I Can't Believe It's Not Better: Challenges in Applied Deep Learning" at ICLR 2025 Workshops*, volume 296 of *Proceedings of Machine Learning Research*, pages 141–150. PMLR, 2025.

[21] Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social Networks*, 5(2):109–137, 1983.

[22] Brian Karrer and Mark EJ Newman. Stochastic blockmodels and community structure in networks. *Physical Review E*, 83(1):016107, 2011.

[23] Tiago P Peixoto. Hierarchical block structures and high-resolution model selection in large networks. *Physical Review X*, 4(1):011047, 2014.

[24] Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268–276, 2018.

[25] Christopher Morris, Nils M Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. TUDataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663*, 2020.

[26] Scott Freitas, Yuxiao Dong, Joshua Neil, and Duen Horng Chau. A large-scale database for graph representation learning. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.

[27] Vincent A Traag, Ludo Waltman, and Nees Jan Van Eck. From Louvain to Leiden: guaranteeing well-connected communities. *Scientific Reports*, 9(1):5233, 2019.

[28] Cedric Sanders, Andreas Roth, and Thomas Liebig. Curvature-based pooling within graph neural networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 471–485, 2023.

[29] Enxhell Luzhnica, Ben Day, and Pietro Lio. Clique pooling for graph classification. *arXiv preprint arXiv:1904.00374*, 2019.

[30] Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. In *Advances in Neural Information Processing Systems*, volume 35, pages 14501–14515, 2022.

[31] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *International Conference on Machine Learning*, pages 3734–3743, 2019.

[32] Ekagra Ranjan, Soumya Sanyal, and Partha Talukdar. ASAP: Adaptive structure aware pooling for learning hierarchical graph representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5470–5477, 2020.

[33] Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. Spectral Clustering with Graph Neural Networks for Graph Pooling. In *International Conference on Machine Learning*, 2020.

[34] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical Graph Representation Learning with Differentiable Pooling. In *Advances in Neural Information Processing Systems*, 2018.

[35] Yash Deshpande, Subhabrata Sen, Andrea Montanari, and Elchanan Mossel. Contextual stochastic block models. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[36] Timo Stoll, Chendi Qian, Ben Finkelshtein, Ali Parviz, Darius Weber, Fabrizio Frasca, Hadar Shavit, Antoine Siraudin, Arman Mielke, Marie Anastacio, Erik Müller, Maya Bechler-Speicher, Michael Bronstein, Mikhail Galkin, Holger Hoos, Mathias Niepert, Bryan Perozzi, Jan Tönshoff, and Christopher Morris. GraphBench: Next-generation graph learning benchmarking. *arXiv preprint arXiv:2512.04475*, 2025.

[37] Paul Erdős and Alfréd Rényi. On random graphs I. *Publicationes Mathematicae Debrecen*, 6: 290–297, 1959.

[38] Ke Xu, Frédéric Boussemart, Fred Hemery, and Christophe Lecoutre. Random constraint satisfaction: Easy generation of hard (satisfiable) instances. *Artificial Intelligence*, 171(8): 514–534, 2007.

[39] Zhuo Xu, Lu Bai, Lixin Cui, Ming Li, Hangyuan Du, Ziyu Lyu, Yue Wang, and Edwin R Hancock. SSHPool: The separated subgraph-based hierarchical pooling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 27405–27413, 2026.

[40] Xiaoxin He, Bryan Hooi, Thomas Laurent, Adam Perold, Yann LeCun, and Xavier Bresson. A generalization of ViT/MLP-Mixer to graphs. In *International Conference on Machine Learning*, pages 12724–12745, 2023.

[41] Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. Hypergraph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3558–3565, 2019.

[42] Iulia Duta and Pietro Lio. Sphinx: Structural prediction using hypergraph inference network. In *International Conference on Machine Learning*, pages 14884–14901, 2025.

[43] Sunwoo Kim, Soo Yong Lee, Yue Gao, Alessia Antelmi, Mirko Polato, and Kijung Shin. A survey on hypergraph neural networks: An in-depth and step-by-step guide. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 6534–6544, 2024.

[44] Yanbo Wang and Muhan Zhang. An empirical study of realized GNN expressiveness. In *International Conference on Machine Learning*, pages 52134–52155, 2024.

[45] Vikraman Arvind, Frank Fuhlbrück, Johannes Köbler, and Oleg Verbitsky. On Weisfeiler-Leman invariance: Subgraph counts and related graph properties. *Journal of Computer and System Sciences*, 113:42–59, 2020.

[46] Hongyang Gao and Shuiwang Ji. Graph U-Nets. In *International Conference on Machine Learning*, pages 2083–2092, 2019.

[47] Boris Knyazev, Graham W Taylor, and Mohamed Amer. Understanding attention and generalization in graph neural networks. In *Advances in Neural Information Processing Systems*, 2019.

[48] Cheng Shi, Liming Pan, Hong Hu, and Ivan Dokmanić. Homophily modulates double descent generalization in graph convolution networks. *Proceedings of the National Academy of Sciences*, 121(8):e2309504121, 2024.

[49] Nikil Wale, Ian A Watson, and George Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14:347–375, 2008.

[50] Vijay Prakash Dwivedi, Chaitanya K Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *Journal of Machine Learning Research*, 24(43):1–48, 2023.

[51] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[52] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference*, pages 11–15, 2008.

[53] Tiago P. Peixoto. The graph-tool Python library. *figshare*, 2014. doi: 10.6084/m9.figshare.1164194.

[54] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[55] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[56] Tianle Cai, Shengjie Luo, Keyulu Xu, Di He, Tie-yan Liu, and Liwei Wang. GraphNorm: A principled approach to accelerating graph neural network training. In *International Conference on Machine Learning*, pages 1204–1215, 2021.

[57] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

[58] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with performers. In *International Conference on Learning Representations*, 2021.

[59] Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph neural networks with learnable structural and positional representations. In *International Conference on Learning Representations*, 2022.

[60] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *International Conference on Learning Representations*, 2020.

[61] Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using PageRank vectors. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 475–486. IEEE, 2006.

Appendix Overview

The appendix contains the following sections:

- **Appendix A: Proofs for Theoretical Analysis.** We present formal proofs and additional examples for the theoretical analysis presented in Section 3.2.
- **Appendix B: Realized Expressivity on BREC.** We validate our theoretical results from Section 3.2 on a benchmark testing model expressivity [44].
- **Appendix C: Additional Results on Synthetic Datasets.** We discuss additional results on our synthetic datasets for learned pooling schemes popular in practice and we extend our synthetic datasets with meaningful node features.
- **Appendix D: Datasets.** We present details on the datasets used in our experimental evaluation. We first explain the exact generation procedure to generate our synthetic benchmarks. Then, we give a short overview and summary statistics across datasets.
- **Appendix E: Experimental Details.** We report the necessary details to fully reproduce our experiments. We also briefly discuss the impact of hyperparameters on real-world datasets.
- **Appendix F: Results for Additional Subgraph Policies.** We present and briefly discuss additional results on the synthetic and real-world datasets for three more subgraph policies. These are not included in the main text as these are mainly devised by us, and therefore do not have a strong basis in literature.

A Proofs for Theoretical Analysis

Assumptions. Throughout this appendix, we assume that all message passing layers are permutation-equivariant and at most 1-WL expressive, as is standard for MPNNs such as GIN [4]. We further assume that the within-subgraph message passing layer used by IP-GNN can realize the identity map, for instance via residual connections. Finally, whenever pooling reduces node representations to supernode representations, we assume that the reduction operator RED is injective on multisets of node embeddings, so that it does not destroy distinctions already present in the pooled clusters [11, 12]. For PSA-GNN, we also assume that the final subgraph-level or supergraph-level readout is chosen injectively whenever needed. All expressivity comparisons are understood in the sense of Definition 1.

A.1 Proof of Proposition 3

We show first that IP-GNN is at least as expressive as Pool-GNN, and then give a graph pair for which it is strictly more expressive.

At least as expressive. Let M_{pool} be any Pool-GNN under a fixed hard partition $\mathcal{C} = \{A_1, \ldots, A_m\}$. Consider the corresponding IP-GNN under the same partition. By assumption, the additional within-subgraph message passing layer can realize the identity map. Hence we may choose its parameters so that it leaves all node representations unchanged before the subsequent RED-CON pooling step. With this choice, IP-GNN reproduces exactly the same computation as M_{pool} on every input graph. Therefore, for every Pool-GNN there exists an IP-GNN with identical outputs, and IP-GNN is at least as expressive as Pool-GNN.

Strictly more expressive. It remains to show that there exists a partition $\widetilde{\mathcal{C}}$ and a pair of graphs that can be distinguished by IP-GNN but not by Pool-GNN. Consider two graphs G and H whose node sets are partitioned into two clusters A and B of equal size, with $G[A] \cong P_4$, $G[B] \cong P_4$, and $H[A] \cong K_{1,3}$, $H[B] \cong K_{1,3}$. Add inter-cluster edges so that both full graphs are regular and so that the coarsened graph obtained after pooling is identical in both cases.

A model in Pool-GNN immediately reduces each cluster to a supernode. Since the pooled graph is the same for G and H, and since the pre-pooling message passing stage cannot distinguish the regular unlabeled input graphs, Pool-GNN cannot separate the pair.

By contrast, IP-GNN performs an additional message passing step inside each induced subgraph before reduction. A 1-WL-bounded MPNN can distinguish P_4 from $K_{1,3}$ via their different internal degree structure, and the injective reduction preserves this distinction at the supernode level. Hence IP-GNN distinguishes the pair, establishing strict separation. □

Additional example. The above argument is not limited to hand-crafted partitions. In practice, structure-based pooling policies can induce subgraphs whose internal structure is informative even when the final coarsened graph is not. Figure 8 gives an additional example based on the Shrikhande graph and the 4×4 Rook's graph, a classical pair of non-isomorphic strongly regular graphs with parameters $\mathrm{SRG}(16, 6, 2, 2)$ [45]. Under an empirical Leiden-based partition, the induced subgraphs expose different internal degree signatures. These are accessible to IP-GNN and PSA-GNN, but are discarded by immediate coarsening.

A.2 Proof of Proposition 4

We prove the two claims in turn. First, PSA-GNN with supergraph readout subsumes IP-GNN under any fixed policy π. Second, there exists a policy π such that the padded version of PSA-GNN is strictly more expressive than IP-GNN under the base policy.

We equip PSA-GNN with a supergraph readout only to match the post-pooling interface of IP-GNN in the expressivity argument. In practice, we use a DeepSets-style readout by default, since our hypothesis is that the relevant hierarchical signal is already captured by the processed pooling-induced substructures and their coupling through the original graph branch.

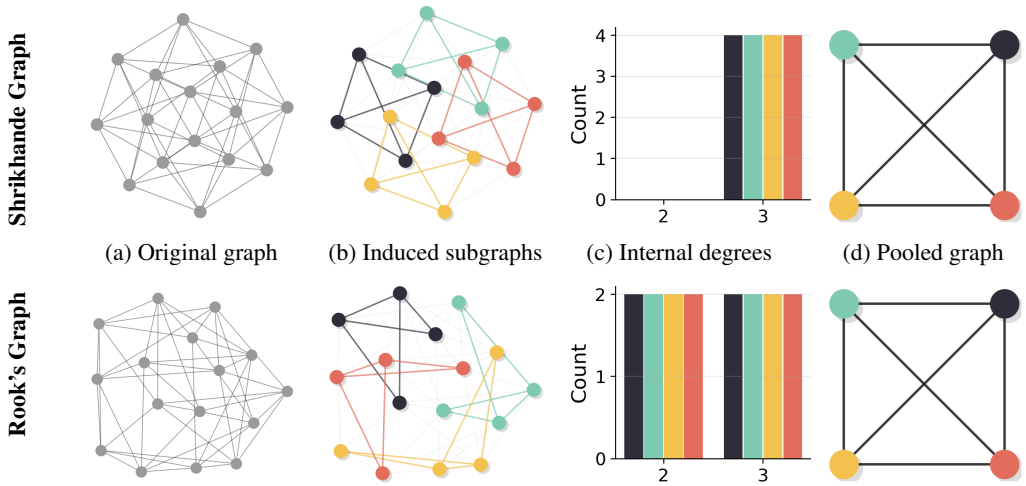


Figure 8: Additional example for Proposition 3. The Shrikhande graph and the 4×4 Rook graph are not distinguished by immediate coarsening under this Leiden partition, but their pooling-induced subgraphs exhibit different internal degree signatures that can be exploited by IP-GNN or PSA-GNN.

(1) At least as expressive as IP-GNN. Fix any subgraph policy π. Recall that one PSA-GNN layer updates subgraph-node features as

$$X_i' \;=\; L^{\mathrm{sg}}(A_i, X_i) \;+\; \big(L^{\mathrm{graph}}(A, \bar{X})\big)_{\pi_i},$$

where L^{sg} acts inside each selected subgraph and L^{graph} acts on the original graph after scatter-aggregating subgraph-node features back to node space.

Consider an IP-GNN with a single pooling step under the same policy π. Its computation can be decomposed into three stages: (i) message passing on the original graph before pooling, (ii) message passing inside the induced subgraphs before reduction, and (iii) message passing on the pooled supergraph after $RED\text{-}CON$.

A PSA-GNN with supergraph readout can emulate these stages sequentially:

- Pre-pooling global message passing is recovered by setting $L^{\mathrm{sg}} = 0$ and choosing L^{graph} to match the corresponding original-graph layers.
- Within-subgraph message passing is recovered by setting $L^{\mathrm{graph}} = 0$ and choosing L^{sg} to match the intra-subgraph layers.
- Since both models use the same policy π and the same $RED\text{-}CON$ mechanism, they induce the same pooled graph. The post-pooling computation of IP-GNN can therefore be reproduced by replacing the DeepSets readout of PSA-GNN with a supergraph readout of the same depth.

Thus every computation realizable by IP-GNN under π can also be realized by PSA-GNN under π. Hence PSA-GNN is always at least as expressive as IP-GNN.

(2) Strict improvement under padded policies. Having shown that PSA-GNN subsumes IP-GNN under π, we now give an example of a pair of graphs that can be distinguished by PSA-GNN but not IP-GNN under a fixed pooling policy. Consider the two 4-regular graphs G and H in Figure 9. Both graphs have vertex set $V = \{1, \ldots, 8\}$ and the same hard partition $A = \{1, 2, 3, 4\}, B = \{5, 6, 7, 8\}$, where the induced subgraphs on A and B are identical diamonds $D = K_4 - e$. G and H are indistinguishable by 1-WL from constant node features but differ in their crossing edges as shown.

IP-GNN cannot distinguish this pair. Under the policy π, IP-GNN sees the same multiset of induced subgraphs, namely $\{D, D\}$, in both graphs. Moreover, the coarsened graph is the same in both cases: two supernodes A and B connected by six crossing edges. Therefore IP-GNN cannot distinguish G and H under this policy.

PSA-GNN distinguishes this pair after two computational layers using $\hat{\pi}$ with core/halo node markings. We focus on the padded A-subgraph. In both graphs, the core is the same diamond on nodes $\{1, 2, 3, 4\}$, while the halo nodes are $\{5, 6, 7, 8\}$. After one 1-WL refinement step on the padded subgraph, the halo nodes split into two types: nodes 5 and 7 have two core neighbors, while nodes 6 and 8 have one core neighbor. In G, the two low-internal-degree core nodes receive different halo-neighborhood multisets $1 : \{6, 8\}$ and $3 : \{5, 7\}$. Thus node 1 sees two halo nodes of padded degree 1, while node 3 sees two halo nodes of padded degree 2. In H, however, $1 : \{8, 7\}$ and $3 : \{6, 5\}$, so nodes 1 and 3 each see one halo node of padded degree 1 and one halo node of padded degree 2. Consequently, after a second 1-WL refinement step, nodes 1 and 3 receive different colors in the padded A-subgraph of G, but remain indistinguishable in the padded A-subgraph of H.

A two-layer subgraph encoder L^{sg} with a 1-WL-expressive base layer can realize these computations. Together with an injective subgraph readout and an injective permutation-invariant graph-level readout, PSA-GNN distinguishes the padded substructure collections of G and H. Hence padded pooled-substructure processing can be strictly more expressive than unpadded intra-pooling processing. □

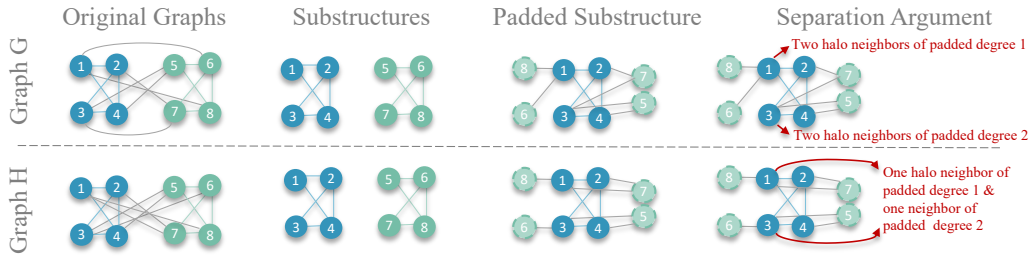


Figure 9: Strict-separation example for padded pooled-substructure processing. The two original graphs G and H are both 4-regular and use the same hard partition into two diamond subgraphs. Under the unpadded policy π, both graphs yield the same subgraphs and coarsened graph with six crossing edges, so IP-GNN cannot distinguish them. Under the padded policy $\hat{\pi}$, however, the A-subgraphs expose different crossing-edge incidence patterns. In G, the core nodes 1 and 3 see different halo-neighbor types and split after two 1-WL refinement steps; while they remain symmetric in H.

B Realized Expressivity on BREC

In this section we verify the expressivity hierarchy from Section 3.2 on a standard realized expressivity benchmark [44]. We present results for Leiden, CliquePool, and CurvPool as well as the Egonet baseline in Figure 10. Clearly, the expected relationship appears in practice: IP-GNN improves over immediate coarsening, while PSA-GNN performs best overall. Experimental details are deferred to Appendix E.

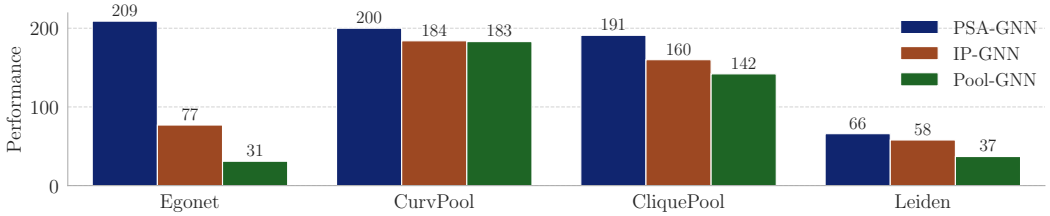


Figure 10: **Number of BREC pairs distinguished by each architecture-policy combination, out of 400 total.** Following the official BREC protocol, we report the best score achieved over three seeds.

C Further Results on Synthetic Datasets

C.1 Hierarchical GNNs on Hierarchy Benchmarks

We further evaluated hierarchical GNNs based on popular pooling schemes that learn cluster assignments based on node features. We distinguish between *dense* and *sparse* methods. Dense methods, such as DiffPool and MinCutPool [34, 33], typically learn a soft assignment matrix $\mathbf{M} \in \mathbb{R}^{|V| \times |V'|}$ that maps all input nodes to supernodes. *Sparse* pooling methods typically learn to assign a score to each node and retain only the top-k nodes, dropping non-selected nodes.

We include DiffPool and MinCut as representative dense pooling schemes [34, 33] as well as TopKPool, SAGPool and ASAPool as sparse variants [46, 47, 31, 32]. As an additional case study, we include randomized variants of ASAPool and SAGPool [10].

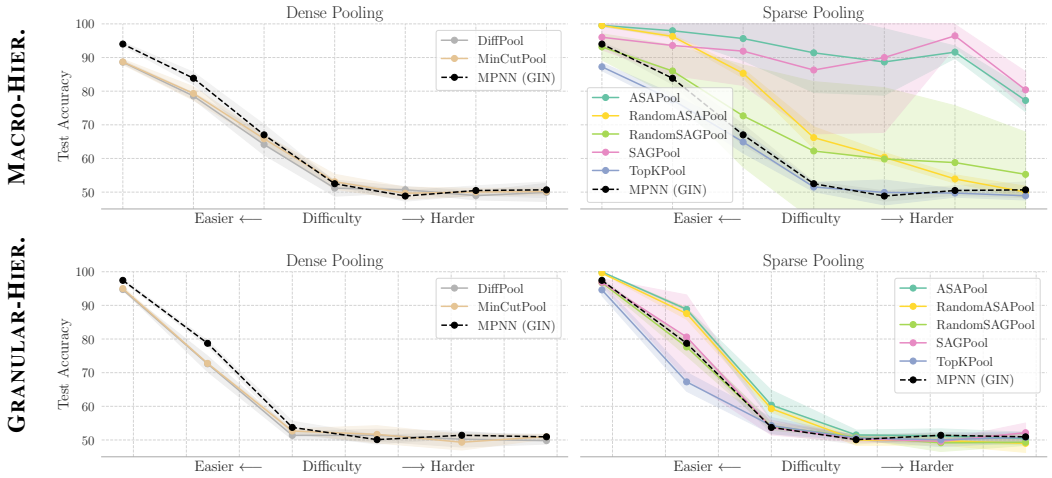


Figure 11: **Results on the synthetic hierarchical benchmarks for learned pooling operators.** We compare a range of popular learned pooling operators as well as their randomized implementations.

On Macro-Hierarchies, all dense pooling schemes fail to improve over the MPNN baseline, likely because the pooling step has little informative node-feature signal to exploit. By contrast, ASAPool and SAGPool, two sparse pooling baselines, achieve best-in-class performance, especially in the harder regimes, albeit at the cost of a higher variance across seeds. Both also clearly outperform their randomized implementations, indicating that they *learn* meaningful hierarchical structure. Furthermore, TopKPool, which implements a similar but simpler coarsening logic, performs substantially worse, suggesting that success in this setting depends on more than sparsity alone. These findings motivate further research on the promise and failure modes of pooling schemes and validate our synthetic approach as a valuable tool to that end.

On Granular-Hierarchies, all learned pooling-based GNNs fail to beat the MPNN baseline. Similarly to Macro-Hierarchies, dense pooling operators might not learn meaningful cluster assignments from graph topology alone. Sparse schemes on the other hand drop nodes, which might not be a suitable inductive bias as solving the task requires recognizing and estimating cluster sizes.

C.2 Contextual Node Features on Hierarchy Benchmarks

The original experiments on our synthetic datasets were conducted with constant node features as these benchmarks are designed to isolate hierarchical structural reasoning. To test whether this design influences our empirical results, we repeated the medium-difficulty experiments with $F = 16$ node features across varying levels of informativeness.

Following the contextual SBM framework [35] and generalizing the feature construction in Eq. 9 of Shi et al. [48] from binary labels to multiple communities, we generate

$$\mathbf{x}_i = \sqrt{\frac{\mu}{N}}\, \mathbf{u}_{c_i} + \boldsymbol{\xi}_i, \tag{4}$$

Table 3: **Impact of contextual node features on synthetic hierarchy benchmarks.** Test accuracy (mean $\pm$ standard deviation) on the medium-difficulty GRANULAR-HIERARCHIES and MACRO-HIERARCHIES datasets with $F = 16$ contextual node features. The gap denotes the accuracy difference in percentage points between PSA-GNN and the GIN baseline.

	Granular-Hierarchies			Macro-Hierarchies		
Feature Level	**PSA-GNN (Leiden)**	**MPNN (GIN)**	**Gap**	**PSA-GNN (Leiden)**	**MPNN (GIN)**	**Gap**
Featureless	95.70 ± 0.56	53.73 ± 1.84	+41.97	99.06 ± 0.36	52.50 ± 0.64	+46.56
$\mu = 1$	95.10 ± 0.36	49.63 ± 2.15	+45.47	99.37 ± 0.31	49.47 ± 1.00	+49.90
$\mu = 5$	95.33 ± 0.47	57.07 ± 2.74	+38.26	99.23 ± 0.25	51.40 ± 2.49	+47.83
$\mu = 10$	95.97 ± 0.47	66.20 ± 1.32	+29.77	99.53 ± 0.06	49.60 ± 2.55	+49.93
$\mu = 50$	97.40 ± 0.90	92.67 ± 0.60	+4.73	99.40 ± 0.26	50.30 ± 2.95	+49.10

where c_i is the lowest-level community of node i, $u_c \sim \mathcal{N}(0, I_F/F)$ are random community prototypes, and $\xi_i \sim \mathcal{N}(0, I_F/F)$. The prototypes are resampled independently for every graph, so they provide noisy community information without directly encoding the graph-level label. Larger μ corresponds to more informative features.

On GRANULAR-HIERARCHIES, increasingly informative features help GIN identify the lowest-level communities and estimate their sizes, narrowing the performance gap. On MACRO-HIERARCHIES, however, information about these communities alone is insufficient to recover their higher-level ring or bipartite organization, and GIN remains near chance. These results suggest that informative node features can reduce the need for structural reasoning when closely aligned with the prediction target as expected, but do not eliminate PSA-GNN's advantage when the task requires reasoning about relationships between communities.

D Dataset Details

Here, we provide additional details on all datasets used in our experiments.

D.1 Synthetic Datasets

This appendix section provides details of the exact construction procedure of graphs in both MACRO-HIERARCHIES and GRANULAR-HIERARCHIES. We generate 10,000 graphs for both benchmarks for various difficulty levels. All nodes are endowed with constant features, unless otherwise specified.

MACRO-HIERARCHIES. This dataset is generated by a two-level stochastic block model with M macro-communities, each consisting of K micro-communities of size S. The class signal is encoded in the connectivity pattern between macro-communities: at the macro level, they are arranged either as a ring or as a bipartite graph, as illustrated in Figure 3.

Concretely, we sample $M \in \{6, 7, 8\}$ and $K \in \{3, 4, 5\}$, and then choose $S \in \{6, 8, 10, 12\}$ such that $MKS \approx 200$. Nodes within the same micro-community are connected with probability $p_{\text{micro}} = 0.35$, while nodes from different micro-communities within the same macro-community are connected with probability $p_{\text{macro}} = 0.12$. Connections between nodes in different macro-communities are then sampled according to the class pattern (ring or bipartite). If two macro-communities are adjacent in the underlying pattern, edges between them are generated with probability $p_{\text{signal}} = p_{\text{base}} + \delta$, where $p_{\text{base}} = 0.01$ is a small background-noise term and $\delta \in (0.002, 0.014)$ controls the strength of the class signal. For pairs of macro-communities that are not connected in the underlying pattern, edges are sampled with probability p_{outside}. This quantity is chosen so that the expected node degree is matched across labels, preventing trivial separation by simple graph statistics. Varying δ therefore controls the difficulty of the task.

GRANULAR-HIERARCHIES. In our second dataset, the class signal is encoded in the relative sizes of the clusters. Graphs with $y = 0$ consist of equally sized clusters, whereas graphs with $y = 1$ contain clusters of unequal size, as illustrated in Figure 3.

As before, we sample the number of clusters as $M \in \{6, 7, 8\}$ and fix the total number of nodes to $N = 200$. We further set a target expected degree of $d_{\text{target}} = 5$. For class $y = 1$, two clusters are

chosen to be four times larger than the remaining ones, whereas for class $y = 0$, all clusters have equal size. Edges between nodes in different clusters are sampled with probability $p_{\text{global}} \in (0.002, 0.022)$, which acts as background noise and therefore controls the difficulty of the task. Within-cluster edges are then sampled with probability p_{within}, chosen such that the expected node degree remains close to d_{target}. This prevents trivial separation based on the total number of edges.

D.2 Dataset Statistics

Table 4 presents summary statistics of the main datasets used in our experiments.

Table 4: **Summary statistics of the benchmark datasets.** Dataset-level statistics are reported as mean $\pm$ standard deviation over graphs where applicable.

Dataset	Task	# Graphs	# Labels	Node feat. dim	Edge feat. dim	Avg. # Nodes	Avg. # Edges	Avg. Degree
Hierarchical Structure								
COLLAB	Classification	5,000	3	0	0	74 ± 62	$2{,}457 \pm 6{,}439$	37.4 ± 44.0
REDDIT-5K	Classification	5,000	5	0	0	509 ± 453	595 ± 567	2.3 ± 0.2
REDDIT-12K	Classification	11,929	11	0	0	391 ± 429	457 ± 518	2.3 ± 0.2
MalNet	Classification	5,000	5	0	0	$1{,}522 \pm 1{,}650$	$2{,}859 \pm 3{,}205$	3.0 ± 1.2
CO-RB	Regression	10,000	1	0	0	241 ± 28	$4{,}497 \pm 1{,}504$	36.8 ± 9.9
Flat Structure								
NCI1	Classification	4,110	2	37	0	30 ± 14	32 ± 15	2.2 ± 0.1
ZINC	Regression	12,000	1	1	1	23 ± 4	25 ± 5	2.1 ± 0.1
CO-ER	Regression	10,000	1	0	0	250 ± 29	$4{,}721 \pm 1{,}090$	37.3 ± 4.4
Synthetic, structure-controlled datasets								
Macro-Hierarchies	Classification	10,000	2	0	0	237 ± 46	942 ± 322	7.8 ± 1.2
Granular-Hierarchies	Classification	10,000	2	0	0	200 ± 0	600 ± 23	6.0 ± 0.2

Our synthetic datasets are introduced in Section D.1. We briefly describe the real-world benchmarks in the domains of malware analysis, social interaction, scientific collaboration, and molecular property prediction. **MalNet-Tiny** contains function call graphs of malicious software and requires predicting the malware family, providing a large-scale and structurally diverse benchmark [26]. **REDDIT-5K** and **REDDIT-MULTI-12K** model Reddit discussion threads as user-interaction graphs and require predicting the originating subreddit [25]. **COLLAB** contains collaboration ego-networks of scientists, with the goal of classifying the research domain from graph topology [25]. **NCI1** is a molecular benchmark in which the task is to predict anti-cancer activity from atom–bond graphs [25, 49]. **ZINC** is a popular molecular regression benchmark based on a 12K subset of the ZINC molecular graph dataset, where the task is to predict constrained solubility from atom–bond graphs [24, 50]. Finally, **CO-ER** and **CO-RB** are graph-level regression benchmarks from GraphBench [36], where the task is to predict the maximum independent set objective on Erdős–Rényi random graphs [37] and clique-structured RB graphs [38], respectively.

E Experimental Details

We conduct all experiments with PyTorch Geometric [51], relying on NetworkX [52] and graph-tool [53] for graph-specific algorithms. Computations are conducted on a single NVIDIA A100 GPU with 80GB VRAM [2].

Model Architecture. We combine all of our processing layers with a standardized base model architecture. Initial node features undergo a linear projection to the embedding space. Subgraph node markings (if used) are also projected and summed with the node embeddings. These features, defined either in node space or in subgraph-node space, are then transformed sequentially by the corresponding layer using a GIN with 2-layer MLPs. After each convolution, we apply batch normalization [54], ReLU activation, dropout, and a residual connection. We use a 2-layer MLP to transform aggregated subgraph features as in Eq. (2) and a 3-layer MLP to produce the final graph-level predictions, with all MLPs containing batch normalization, ReLU, and dropout. Unless otherwise indicated, we use mean aggregation throughout for stability, including for graph-level and supernode-level readout as well as for subgraph feature aggregation in Eq. (1) in PSA-GNN.

[2] A single model training run takes 10 to 30 minutes on synthetic and around 20 to 70 minutes on real-world datasets.

Experimental Setup. All models are matched in depth and overall parameter count, where appropriate. Generally, our goal is not to maximize absolute performance, but to compare the three model architectures in a controlled setting. For all deterministic pooling models, we use a single pooling step, ensuring a like-for-like comparison with PSA-GNN and allowing subgraph assignments to be precomputed. As discussed in Section 5, we compare all models under the same base policy π to isolate the effect of architecture, with PSA-GNN using the padded policy $\hat{\pi}$. Hyperparameters are selected based on validation performance. Across different seeds, we use a random 80%-10%-10% train-validation-test split, or a fixed split when one is provided.

Implementation of Subgraph Policies. We do not tune subgraph-policy hyperparameters and instead use a small set of fixed implementation defaults. Leiden is run with its default resolution parameter of 1.0. For CurvPool, we threshold Balanced Forman Curvature at the 0.8 quantile (which we compared against mean-thresholding), retain edges above this threshold, and merge overlapping selected edges into core clusters; uncovered nodes are subsequently added through a fallback coverage step. For CliquePool, we compute maximal cliques, then resolve overlaps by keeping each node only in its largest incident clique(s), iterating this pruning until stable. In practice, we add node markings beyond the core/halo indicators whenever they are naturally available from the extraction procedure.

E.1 BREC

We largely follow the original BREC protocol [44]. For each pair of graphs, we train a model by optimizing the cosine embedding loss for a maximum of 40 epochs using Adam [55] with learning rate 10^{-4} and weight decay 10^{-4}. We also replace all batch normalization layers with GraphNorm layers [56] due to numerical stability issues. Finally, we use a sum-based graph-level readout.

After some heuristic tuning, we settled on 8 layers throughout. Table 5 shows how these layers are allocated for the Pool-GNN and IP-GNN models. With this setup, our Pool-GNN baseline already achieves stronger results than previous empirical investigations into expressive graph pooling (see Table 1 in [11]), supporting our hyperparameter choices and the use of a single pooling step. Intuitively, if a first pooling step does not already improve separation on BREC, additional pooling steps are unlikely to provide additional help.

Table 5: **Layer allocations across datasets and model families.** For IP-GNN and Pool-GNN, layers are split into *pre*, *within*, and *post*; for **Hierarchical**, they are split into *pre*, *between*, and *post*. PSA-GNN and any MPNN baselines always use the full stack, i.e., total depth. *between* layers are multiplied by the number of pooling steps.

	Total Depth	**IP-GNN**			**Pool-GNN**			**Hierarchical**		
Dataset		*pre*	*within*	*post*	*pre*	*within*	*post*	*pre*	*between*	*post*
Hierarchical Structure										
COLLAB	**6**	2	2	2	3	0	3	2	2x2	0
REDDIT-12K	**6**	2	2	2	3	0	3	2	2x2	0
REDDIT-5K	**6**	2	2	2	3	0	3	2	2x2	0
MalNet	**6**	2	2	2	3	0	3	2	2x2	0
CO-RB	**6**	2	2	2	3	0	3	–	–	–
Flat Structure										
NCI1	**6**	2	2	2	3	0	3	2	2x2	0
ZINC	**8**	4	2	2	5	0	3	4	2x2	0
CO-ER	**6**	2	2	2	3	0	3	–	–	–
Synthetic, structure-controlled datasets										
BREC	**8**	2	3	3	4	0	4	–	–	–
Macro-Hierarchies	**8**	4	2	2	5	0	3	4	2x2	2
Granular-Hierarchies	**8**	4	2	2	5	0	3	4	2x2	2

E.2 Synthetic Datasets

We performed a small hyperparameter sweep over the number of layers $\{8, 12\}$ and embedding size $\{64, 128\}$, evaluating each setting on two seeds. We find that larger models achieve slightly better absolute results, but the relative ordering of the models and the performance gaps between them remain largely unchanged (see Figure 5). We therefore settle on 8 layers with an embedding size of 64. To roughly match the number of parameters, we reduce the embedding size of PSA-GNN to 48. Exact layer allocations for each model are presented in Table 5. We use AdamW [57] for optimization. All other optimization and model hyperparameters are detailed in Table 6.

The GPS [30] baseline uses a GIN layer as the local message-passing component and employs Performer attention [58] for scalability. We set the embedding size to 18 to closely match the parameter budget of the other baselines. We further add Random Walk Positional/Structural Encodings [59] with length 16, a standard structural encoding for graph transformers. In preliminary experiments, GPS without positional/structural encodings performed substantially worse on our synthetic tasks.

Table 6: **Dataset-specific training and architecture hyperparameters.** Key hyperparameters used for each benchmark dataset. For BREC, no patience-based early stopping is used; instead, training stops once the loss falls below 0.2 or after 40 epochs.

Dataset	# Layers	Max Epochs	Batch Size	Learning Rate	Weight Decay	Layers Dropout	Head Dropout	Early stop Patience	# Seeds
Hierarchical Structure									
COLLAB	6	300	32	10^{-3}	0	0.05	0.10	50	5
REDDIT-12K	6	300	16	10^{-3}	0	0.05	0.10	50	5
REDDIT-5K	6	300	16	10^{-3}	0	0.05	0.10	50	5
MalNet	6	300	16	10^{-3}	0	0.05	0.10	50	5
CO-RB	6	300	32	10^{-3}	10^{-5}	0.00	0.10	50	3
Flat Structure									
NCI1	6	300	32	10^{-3}	0	0.00	0.10	50	10
ZINC	8	500	64	10^{-3}	10^{-5}	0.00	0.00	100	3
CO-ER	6	300	32	10^{-3}	10^{-5}	0.00	0.10	50	3
Synthetic, structure-controlled datasets									
BREC	8	40	16	10^{-4}	10^{-4}	0.00	0.00	–	3
Macro-Hierarchies	8	250	128	5×10^{-4}	10^{-5}	0.00	0.00	50	5
Granular-Hierarchies	8	250	128	5×10^{-4}	10^{-5}	0.00	0.00	50	3

We also tuned the number of pooling steps in $\{1, 2\}$ and the pooling strength using either ratios $\{0.1, 0.5\}$ or a fixed target size of 24 nodes, which roughly corresponds to the number of micro-communities on the MACRO-HIERARCHIES dataset. We find that a pooling ratio of 0.5 consistently performs best. For all methods except TopKPool, a single pooling step is optimal.

E.3 Real-World Datasets

For the real-world benchmarks, we use 6 layers on all datasets except ZINC, where 8 layers is more appropriate. Model-specific allocations of layers are reported in Table 5. All other hyperparameters used are detailed in Table 6. Given this model specification, we set the embedding dimension such that each model fits within a 500K parameter budget. As ZINC uses edge features, we first embed edge features and subsequently employ the GINE layer [60] for processing. For datasets without node features, we use constant scalars.

We tune the number of pooling steps in $\{1, 2\}$ for all hierarchical GNNs and find that using two pooling steps is generally beneficial. We further ablate the use of subgraph node markings and the edge-covering policy $\hat{\pi}$. Table 7 presents the result of this experiment. The base model always includes node markings and the padded policy. The impact on performance depends on the dataset and subgraph-policy combination. In most cases, however, the effect is small and remains within one standard deviation of the base model, with the clearest exception being clique-based policies on COLLAB.

Finally, Table 8 reports the average training time per epoch of the real-world datasets NCI1 and REDDIT-12K discussed in Section 5.4. Although the subgraph-based architectures introduce addi-

Table 7: **Padding and node-marking impact for PSA-GNN on real-world datasets.** The base row is the mean-aggregation configuration with node markings enabled and default padding. Italic rows report the change relative to the base. For accuracy metrics, positive values indicate improvement; for ZINC, negative values indicate improvement because lower MAE is better. Models without padding also contain no node markings, as there is no core vs. halo distinction. Empty cells indicate that a configuration was not tested due to computational limitations.

		Hierarchical Structure			Flat Structure	
Model	**Subgraph Policy**	**COLLAB** *Acc. [%] (↑)*	**REDDIT-12K** *Acc. [%] (↑)*	**MalNet** *Acc. [%] (↑)*	**NCI1** *Acc. [%] (↑)*	**ZINC** *MAE (↓)*
PSA-GNN	CliquePool	78.1 ± 1.3	47.0 ± 3.8	90.6 ± 2.4	81.5 ± 2.0	0.181 ± 0.005
	- padding	-4.4	+0.8	+1.0	-0.6	-0.002
	- node markings	-2.3	+0.8	+1.0	+0.0	+0.010
	CurvPool	76.8 ± 1.5	49.3 ± 1.7	92.0 ± 1.9	81.2 ± 1.6	0.175 ± 0.005
	- padding	-0.5	-0.3	-1.2	-0.8	+0.003
	- node markings	+0.5	+0.8	-0.7	-0.8	-0.001
	Leiden	76.4 ± 2.2	50.0 ± 1.3	90.9 ± 1.8	77.8 ± 2.3	0.215 ± 0.012
	- padding	-1.4	-0.8	+0.2	+1.0	-0.006
	- node markings	-1.8	-0.0	-0.4	+2.1	+0.002

tional computational overhead relative to the MPNN baseline, their runtime remains within the same order of magnitude on both datasets, consistent with the theoretical complexity analysis.

Table 8: **Training time per epoch on NCI1 and REDDIT-12K.** Mean training time in seconds per epoch (± standard deviation) for the datasets considered in Figure 7.

Model	**Subgraph Policy**	**NCI1**	**REDDIT-12K**
PSA-GNN	CliquePool	2.47 ± 0.12	16.11 ± 0.89
	CurvPool	2.75 ± 0.18	18.95 ± 2.88
	Leiden	2.45 ± 0.09	17.11 ± 0.75
IP-GNN	CliquePool	2.15 ± 0.17	15.04 ± 2.62
	CurvPool	2.08 ± 0.10	15.05 ± 1.30
	Leiden	2.04 ± 0.04	13.79 ± 1.49
Pool-GNN	CliquePool	2.08 ± 0.15	13.41 ± 1.92
	CurvPool	2.02 ± 0.12	14.68 ± 2.37
	Leiden	1.99 ± 0.05	14.06 ± 0.96
MPNN (GIN)	–	1.16 ± 0.05	9.16 ± 0.54
Egonet	–	2.52 ± 0.16	–

F Additional Results for Other Policies

There are not many widely used structure-based pooling schemes in the graph pooling literature [11]. To expand our evaluation, we therefore repeat our experiments with three additional policies described below.

- **CliquePool-Maximal** is our overlapping variant of CliquePool, in which we keep all maximal cliques as subgraphs instead of resolving overlaps. In the padded version, we add any missing edge to every incident clique-subgraph rather than performing a full 1-hop expansion, so the resulting subgraphs are no longer uniform and contain additional structural variation.
- **ACL** applies the Andersen–Chung–Lang localized clustering algorithm [61] iteratively, each time seeding it at the highest-degree node that has not yet been assigned to a cluster. In detail, clusters are extracted from a personalized PageRank sweep and constrained to cover between 0.02 and 0.3 of the graph volume; if no sweep prefix falls in this range, we choose the prefix closest to the target range. For computational reasons, we cap the iterative ACL

procedure at 1000 extracted subgraphs, after which any remaining uncovered nodes are assigned through a rescue step.

- **SeededHops** is our iterative seed-based variant, which forms a 1-hop egonet around the highest-degree node that has not yet been assigned to a cluster, and repeats this process until all nodes are assigned.

Note that the latter two are seed-based policies, where we seed a subgraph extraction logic at some node and repeat until coverage is achieved. These are closer in spirit to the egonet baseline, which extracts 2-hop expansions around *every* node.

These policies also give the opportunity to benchmark the usage of additional subgraph node features. In particular, for seed-based policies we add a root-node indicator together with distance-to-root features. For ACL, we additionally include the normalized personalized PageRank score of each node within the extracted cluster.

Figure 12 reports the results for the BREC dataset. These new policies follow the exact same trend as the structure-based policies, with PSA-GNN outperforming Pool-GNN and IP-GNN occupying the middle ground. CliquePool-Maximal stands out as a particularly successful approach. This could be potentially explained by the increase in overlap across clusters among all maximal cliques. Notably, this more powerful approach only outperforms standard CliquePool when combined with PSA-GNN but not when processed by Pool-GNN, indicating that more powerful policies need more powerful processing approaches.

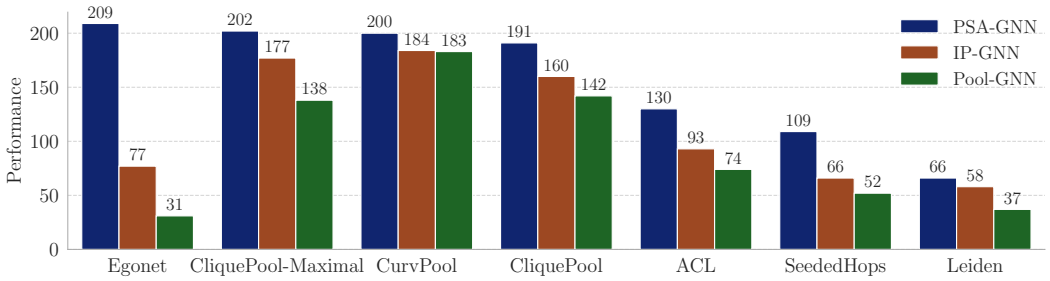


Figure 12: **Number of BREC graph pairs (out of 400) distinguished by each architecture-policy combination.** We report the best score achieved over three seeds.

Next, we repeated the experiment on our synthetic datasets, with results presented in Figure 13, excluding CliquePool-Maximal for computational reasons. Again, the pattern discussed in the main text is clearly confirmed on these additional policies. Note that due to the structure of the dataset, even Egonets arguably group nodes belonging to the same community, as clusters are more densely connected.

Finally, we present an expanded Table 9 reporting results on real-world benchmarks including our new policies. PSA-GNN remains the top performing model. CliquePool-Maximal emerges as a powerful policy, reaching the best result on all hierarchical datasets tested, while remaining computationally feasible.

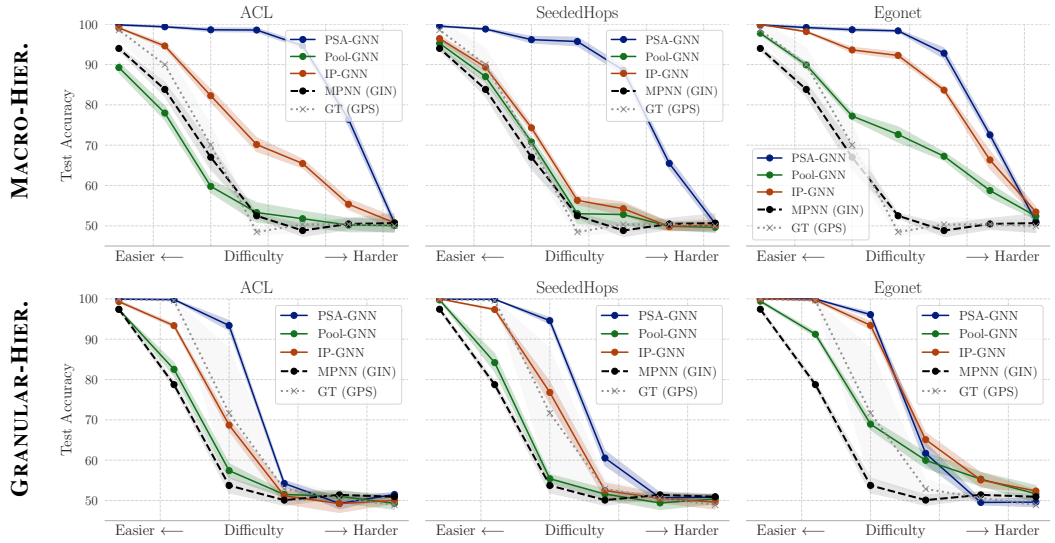


Figure 13: **Results on the synthetic hierarchical benchmarks for additional subgraph policies.** We compare immediate coarsening (Pool-GNN), the within-subgraph baseline (IP-GNN), and the full pooled-subgraph model (PSA-GNN) across three additional subgraph policies.

Table 9: **Results on real-world benchmark datasets.** The best overall result per dataset is shown in **brown**, second best in **purple**. Among PSA-GNN, IP-GNN, and Pool-GNN, the best method for a given subgraph policy π is shown in **bold**. The *Avg. Rank* column reports the average rank of these three architecture levels across all available policy–dataset comparisons.

			Hierarchical Structure			Flat Structure	
Model	**Subgraph Policy**	**Avg. Rank (↓)**	**COLLAB** *Acc. [%] (↑)*	**REDDIT-12K** *Acc. [%] (↑)*	**MalNet-T** *Acc. [%] (↑)*	**NCI1** *Acc. [%] (↑)*	**ZINC** *MAE (↓)*
PSA-GNN	CliquePool	**1.34**	**78.1 ± 1.3**	47.8 ± 1.4	90.6 ± 2.4	**81.5 ± 2.5**	**0.182 ± 0.008**
	CliquePool-max		**78.9 ± 0.7**	**51.2 ± 0.0**	**92.3 ± 1.6**	80.9 ± 1.6	**0.179 ± 0.003**
	CurvPool		76.8 ± 1.5	**50.1 ± 1.7**	**92.0 ± 1.9**	81.2 ± 1.6	**0.177 ± 0.005**
	Leiden		**76.4 ± 2.2**	**50.0 ± 1.3**	**91.1 ± 1.4**	80.0 ± 2.5	0.213 ± 0.007
	ACL		–	–	**92.1 ± 0.9**	**80.9 ± 2.8**	**0.189 ± 0.002**
	SeededHops		72.2 ± 4.6	**50.4 ± 1.4**	**91.4 ± 2.0**	81.1 ± 2.6	**0.184 ± 0.006**
IP-GNN	CliquePool	2.32	75.0 ± 2.3	45.1 ± 2.1	**91.7 ± 1.7**	80.6 ± 2.1	0.183 ± 0.004
	CliquePool-max		78.1 ± 0.4	49.3 ± 0.7	91.6 ± 1.4	80.2 ± 2.4	0.187 ± 0.002
	CurvPool		**77.4 ± 0.9**	42.0 ± 2.5	89.3 ± 2.0	80.3 ± 1.9	0.184 ± 0.005
	Leiden		73.6 ± 3.1	45.6 ± 0.8	90.7 ± 1.1	**80.1 ± 2.6**	**0.186 ± 0.004**
	ACL		–	–	90.5 ± 1.5	80.7 ± 3.6	0.248 ± 0.005
	SeededHops		71.5 ± 5.9	48.8 ± 1.0	89.6 ± 2.3	81.5 ± 1.6	0.209 ± 0.002
Pool-GNN	CliquePool	2.34	70.4 ± 4.1	**48.3 ± 1.7**	90.2 ± 1.6	81.0 ± 2.4	**0.182 ± 0.004**
	CliquePool-max		76.0 ± 2.3	46.1 ± 6.4	90.2 ± 2.3	80.3 ± 2.4	0.181 ± 0.004
	CurvPool		71.1 ± 4.5	47.4 ± 0.4	91.3 ± 1.1	**81.3 ± 2.0**	0.178 ± 0.004
	Leiden		74.1 ± 2.3	44.7 ± 1.6	87.9 ± 1.5	79.9 ± 2.0	0.257 ± 0.013
	ACL		–	–	89.8 ± 2.0	80.8 ± 2.1	0.270 ± 0.005
	SeededHops		**73.2 ± 2.9**	48.4 ± 1.9	90.4 ± 1.6	**81.6 ± 2.0**	0.229 ± 0.003
Hierarchical GNN	ASAPool	–	71.6 ± 5.4	49.5 ± 0.2	88.2 ± 3.1	81.2 ± 2.0	–
	SAGPool	–	70.6 ± 4.2	46.6 ± 1.8	90.1 ± 1.1	80.1 ± 1.9	0.416 ± 0.107
	MinCutPool	–	–	–	–	80.2 ± 2.0	**0.161 ± 0.003**
	DiffPool	–	–	–	–	81.0 ± 1.7	**0.147 ± 0.005**
Baselines	Egonet	–	–	–	91.4 ± 1.4	**81.6 ± 2.4**	0.403 ± 0.060
	MPNN (GIN)	–	74.9 ± 3.7	48.3 ± 2.3	91.0 ± 1.6	80.9 ± 2.5	0.172 ± 0.005

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The main claims made in the abstract and Section 1 accurately reflect the paper's contribution and scope.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We include a dedicated paragraph discussing limitations of our work in the Section 7.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: We provide complete proofs of our theoretical result including all assumptions in the Appendix.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The main text in combination with the additional information supplied in the Appendix enables a full reproduction of our pipeline and experimental results.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We provide a minimal code base to reproduce our results as supplementary material. This also includes code to reproduce our synthetic datasets. All other data used is publicly available.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: We report exhaustive details on data splits, training process, hyperparameters and further experimental details in the Appendix.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: We report standard deviations for our main synthetic result as well as the real-world benchmarks. We further include details on the underlying data distribution in Figures 4 and 6. We omit standard deviations on BREC as the typical training protocol demands reporting the maximal achieved performance over multiple seeds, which usually exhibits very little variation.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We explain hardware with memory configuration as well as typical runtimes for model training in the Appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We strictly adhere to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: We present a new approach to leverage graph pooling for hierarchical modeling in GNNs. Designing powerful GNNs could in principle have an impact down the line. However, this heavily depends on the type of application domain and further progress in the field. As there are no direct societal impacts that would be meaningfully different from existing approaches, we do not think a contingent discussion is necessary.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: We do not release any assets that would warrant safeguarding.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We properly cite and attribute all datasets we used in our experiments.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We introduce two new synthetic, hierarchical datasets that researchers might find useful. These datasets are well-documented in the main text and the appendix and the code to produce them is included as supplementary material.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: No such research was performed in this work.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: No such research was performed in this work.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: Our method does not involve LLMs as an important, non-standard component.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.