GRAPH-DEPENDENT POOLING: ADAPTIVE GRANULARITY IN HIERARCHICAL GRAPH NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph Neural Networks are widely used to learn structural representations of relational data. Hierarchical graph pooling learns multi-level representations by progressively coarsening graphs; however, prior methods rely on predefined cluster numbers or fixed pooling ratios, limiting adaptability to individual graphs. We propose Graph-Dependent Pooling (GDPool), which dynamically determines pooling granularity for each graph. We analyze optimization conflicts among layer-wise objectives through gradient alignment, showing that layer-wise objectives at different pooling levels can send conflicting gradients to shared parameters. GDPool uses self-attention to select graph-specific anchors representing structurally important nodes and constructs coarse graphs through soft node assignment. Experiments on metal-organic framework and molecular graph benchmarks show that GDPool consistently achieves the best predictive performance across all benchmarks. Evaluations using modularity and the effective number of clusters show that GDPool achieves consistently higher modularity than existing methods while the number of active clusters decreases monotonically with depth. Sensitivity analyses confirm stable behavior across hyperparameter and architectural choices.

1 INTRODUCTION

Graph Neural Networks (GNNs) have become a widely used approach for learning from relational data, including molecules, crystal structures, and social networks (Gilmer et al., 2017; Xu et al., 2019). In molecular and materials property prediction, graph-based models represent atoms and bonds as graphs and directly predict graph-level properties (Xie & Grossman, 2018; Chen et al., 2019). Effective graph-level prediction requires aggregating node-level information into a representation of the entire graph. While standard sum or mean readout aggregates all node representations in a single step, graph pooling progressively coarsens the graph to learn structural representations at multiple levels (Ying et al., 2018; Grattarola et al., 2022; Liu et al., 2022). Hierarchical graph pooling has therefore emerged as a key approach for compressing complex graph structures while preserving salient information, with recent work also theoretically analyzing how pooling operations affect the expressive power of GNNs (Bianchi & Lachi, 2023).

Hierarchical graph pooling involves two key design challenges. The first is determining how much to coarsen a graph. Clustering-based methods such as DiffPool, MinCutPool (Bianchi et al., 2020), and MemPool (Khasahmadi et al., 2020) typically require a predefined number of clusters, whereas selection-based methods such as Graph U-Nets (Gao & Ji, 2019), SAGPool (Lee et al., 2019), and ASAPool (Ranjan et al., 2020) rely on a predefined pooling ratio. To relax this fixed granularity, methods such as GMPool (Ko et al., 2023) and BN-Pool (Castellana & Bianchi, 2026) adapt the number of clusters to each input graph, but still require a hyperparameter specifying the maximum number of clusters. Moreover, learning graph-dependent coarsening consistently across multiple pooling levels remains an open challenge. The second challenge is how to learn these levels as a coherent hierarchy. Existing soft-assignment methods apply auxiliary objectives at individual levels (Ying et al., 2018; Bianchi et al., 2020); however, because each higher-level graph is constructed from the output of the preceding pooling operation, its supervision target recursively depends on earlier assignments. Consequently, supervising each level only through its own local objective does not necessarily ensure structural consistency across the hierarchy.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

To address these challenges, we propose Graph-Dependent Pooling (GDPool). GDPool identifies important nodes as graph-specific anchors and dynamically determines pooling granularity without a predefined number of clusters or pooling ratio. Self-attention estimates node importance, selects anchors, and softly assigns nodes to the selected anchors to construct a coarse graph. We further propose hierarchical supervision, which applies pooling objectives to the cumulative assignments so that every level is anchored to the original graph. GDPool thus learns graph-dependent hierarchical representations. We evaluate predictive performance and hierarchy quality using modularity (Newman & Girvan, 2004) for structural cohesion, the effective number of clusters K_{eff} (Zheng & Vedaldi, 2023) for cluster utilization, and gradient alignment (Sener & Koltun, 2018; Yu et al., 2020) among hierarchical objectives. The evaluation quantifies whether GDPool adapts to each graph while maintaining a consistent hierarchy. Our main contributions are as follows.

- **Graph-dependent pooling.** We propose GDPool, which selects important nodes as anchors via self-attention and thereby determines pooling granularity without a predefined number of clusters or pooling ratio. Each cluster is anchored at an actual node, which makes the hierarchy directly interpretable.
- **Gradient conflict analysis and hierarchical supervision.** Measuring the gradient alignment between the auxiliary objectives of different pooling layers, we identify two sources of conflict: layer-wise supervision and a pooling budget that does not fit the graph. We mitigate the first by supervising the cumulative assignment of every level against the original graph, and the second by graph-dependent anchor selection.
- **Empirical validation across domains.** GDPool achieves the best MAE on all five targets of four MOF and molecular benchmarks against nine existing pooling methods, with a monotonically decreasing number of active clusters and consistently higher modularity. GDPool thus forms coherent hierarchies rather than merely reducing the number of nodes.

2 Related work

2.1 Hierarchical graph pooling

Graph pooling can be grouped into three categories: clustering, node selection, and edge contraction (Grattarola et al., 2022). Clustering-based methods learn soft node-to-supernode assignments, including DiffPool, MinCutPool, and MemPool. TopKPool (Gao & Ji, 2019; Cangea et al., 2018; Knyazev et al., 2019), SAGPool (Lee et al., 2019), ASAPool, and MaxCutPool (Abate & Bianchi, 2025) select important nodes or local clusters, while EdgePool (Diehl, 2019) iteratively merges adjacent nodes. Clustering-based pooling retains explicit assignments of original nodes, enabling direct analysis of hierarchical structures and cluster-level properties.

In clustering-based pooling, selecting representative nodes and determining cluster numbers are key design choices. GAT (Velickovic et al., 2018) and graph Transformers (Dwivedi & Bresson, 2020; Ying et al., 2021; Kreuzer et al., 2021; Rampášek et al., 2022) learn local and global node interactions through attention. While SAGPool and ASAPool rely on local scores and a fixed pooling ratio, GDPool scores nodes with graph-wide attention and lets the number of anchors follow each graph's attention distribution. Unlike ASAPool, which further restricts cluster membership to h-hop neighborhoods, GDPool assigns every node to all anchors without locality constraints or additional learnable parameters. More generally, prior pooling methods fix the number of clusters or the pooling ratio in advance, whereas clustering methods determine cluster numbers from data using eigengaps (Von Luxburg, 2007), nonparametric clustering (Kulis & Jordan, 2011), or modularity maximization (Newman, 2006; Blondel et al., 2008; Traag et al., 2019). In graph pooling, HGP-SL (Zhang et al., 2019) and SEP (Wu et al., 2022) consider adaptive structures, and GMPool (Ko et al., 2023) and BN-Pool also adapt cluster numbers to each graph. However, both must set a maximum number of clusters, and GMPool additionally requires $O(N^3)$ eigendecomposition at each level. After pooling, the coarse graph should preserve the original structure while supporting stable learning across multiple levels. Graph coarsening studies have explored multiresolution construction (Defferrard et al., 2016), spectral and cut preservation (Loukas, 2019; Loukas & Vandergheynst, 2018), and learning-based coarsening and scalability (Cai et al., 2021; Huang et al., 2021; Bacciu et al., 2023). These coarsening methods maintain the composed mapping from the original graph to each level, whereas learned pooling methods define their clustering objectives on each coarsened graph sepa-

rately; to our knowledge, applying these objectives to the composed assignment at every level has not been studied. Hierarchical GNNs such as Graph U-Net alternate message passing and pooling, while GDPool uses a pre-norm architecture (Xiong et al., 2020) to stabilize these levels.

2.2 Deep supervision for hierarchical learning

Hierarchical models derive effectiveness from a sequence of intermediate representations, making the supervision of intermediate levels an important design consideration. Deep supervision was introduced to improve the optimization and representation learning of deep networks by imposing auxiliary objectives directly on intermediate representations. Deeply-Supervised Nets (Lee et al., 2015) attach auxiliary classifiers to hidden layers, and GoogLeNet (Szegedy et al., 2015), HED (Xie & Tu, 2015), and UNet++ (Zhou et al., 2018) extend deep supervision to deeper and multi-scale architectures. Hierarchical graph pooling similarly employs layer-wise auxiliary objectives: DiffPool applies link prediction and entropy regularization to the adjacency and assignment at each level, and MinCutPool uses min-cut and orthogonality objectives. In both methods, each objective is defined with respect to the graph at the corresponding level, so the auxiliary objectives are defined locally at each level while sharing parameters.

When multiple auxiliary objectives jointly optimize shared parameters, the relationship among the gradients of the objectives becomes critical. Multi-task learning addresses gradient interactions by formulating training as multi-objective optimization (Sener & Koltun, 2018), adjusting gradient magnitudes or loss weights with GradNorm (Chen et al., 2018), applying uncertainty weighting (Kendall et al., 2018), or directly modifying conflicting gradients with PCGrad (Yu et al., 2020) and CAGrad (Liu et al., 2021). In particular, Du et al. (2018) weight auxiliary losses by the cosine similarity between auxiliary and main-task gradients, showing that auxiliary supervision is not always aligned with the task objective. However, gradient interactions among layer-wise auxiliary objectives in hierarchical graph pooling have received limited attention.

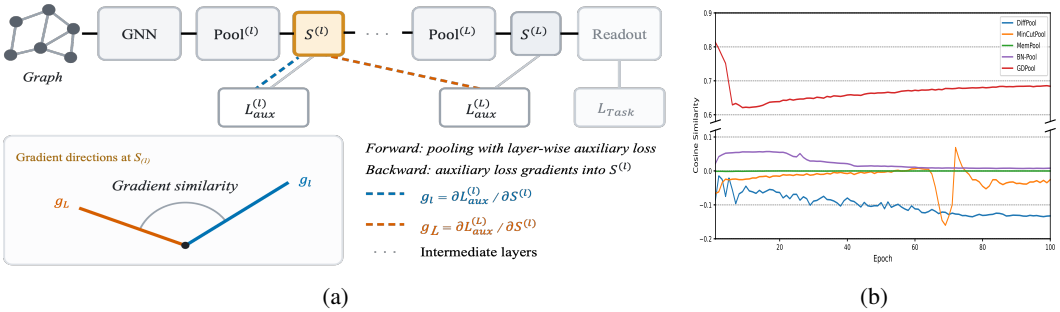
3 Gradient interactions in hierarchical learning

Hierarchical graph learning sequentially constructs graph representations at different levels from a single input graph. Layer-wise supervision assigns a separate objective to the representation at each pooling level, providing a direct learning signal for capturing meaningful information at each level. However, when objectives at different levels share parameters, the gradients of the objectives may point in different directions, so that an update improving the objective at one level degrades the objective at another level. We define the interference between layer-wise objectives as gradient conflict, quantified by the directional alignment of the gradients that the objectives send to shared parameters. Measuring gradient conflict during training, we examine whether layer-wise objectives, each defined on its own coarsened graph, are compatible with learning a structurally coherent hierarchy.

3.1 Gradient conflict in hierarchical learning

Gradient conflict occurs when layer-wise objectives send gradients in opposing directions to shared parameters, so that improving one objective hinders another. To quantify gradient conflict, let $\mathcal{L}_{\text{aux}}^{(l)}$ denote the auxiliary objective at the l-th pooling level. For two levels $l < L$, we take the gradients of both objectives with respect to the lower-level assignment $S^{(l)}$, $g_l = \partial \mathcal{L}_{\text{aux}}^{(l)} / \partial S^{(l)}$ and $g_L = \partial \mathcal{L}_{\text{aux}}^{(L)} / \partial S^{(l)}$ ($l = 1$ in our experiments). Similar gradient directions at levels i and j indicate cooperative optimization, whereas opposing directions indicate that an update for one objective hinders the other. We quantify the directional relationship between two levels by gradient alignment. Positive alignment indicates that the two objectives favor similar update directions, whereas negative alignment signals likely gradient conflict; alignment near zero implies nearly orthogonal gradients with little mutual influence. Figure 1a illustrates the measurement procedure, in which the gradients of the auxiliary objectives at each pooling level are computed and compared at $S^{(l)}$. Unlike a comparison of loss magnitudes, gradient alignment directly reveals the direction in which each layer-wise objective drives the shared parameters. Figure 1b shows the gradient alignment between the auxiliary objectives of DiffPool, MinCutPool, MemPool, and BN-Pool, measured during training. The alignment of every baseline remains below 0.1 throughout training, either staying close

to zero or falling below zero for extended periods. Layer-wise objectives therefore fail to provide cooperative learning signals, either conflicting with or barely influencing one another.



(a) (b)

Figure 1: (a) Cosine similarity used to quantify the alignment between gradients of layer-wise auxiliary objectives (b) Gradient alignment across hierarchy levels during training on the ARC-MOF dataset; the red line denotes GDPool.

3.2 Implications of Gradient Conflict

The observed gradient conflict suggests that applying a separate local objective at each level does not guarantee that the hierarchy is learned coherently. When layer-wise objectives send divergent gradients to shared parameters, learning at one level may remain incompatible with representation learning at another level, even if each objective is optimized individually. Satisfying the layer-wise objectives and preserving the structural relationships across the hierarchy can thus become distinct goals. Rather than assuming that layer-wise supervision alone yields a structurally coherent hierarchy, supervision should explicitly reflect the relationships across levels, since the representations at all levels are derived from a single original graph through successive coarsening. Building on the analysis, we propose hierarchical supervision, which applies existing pooling objectives to the cumulative assignments so that the supervision at each level reflects the structural relationships across levels. Section 5.4 analyzes the effect of hierarchical supervision on GDPool, further examining how gradient conflict across levels changes when a fixed pooling granularity is not enforced.

4 Graph-Dependent Hierarchical Pooling

We propose GDPool, which adaptively determines pooling granularity according to the input graph. GDPool computes context-aware node importance by self-attention over the node representations and selects the most important nodes as graph-specific anchors. A coarse graph is then constructed through soft-assignment of each node to the selected anchors. GDPool thereby learns hierarchical representations whose granularity differs across graphs, without a predefined number of clusters or pooling ratio.

4.1 Context-Aware Anchor Selection

GDPool computes context-aware node importance through self-attention, which accounts for interactions with the other nodes in the graph. Let $G = (X, A)$ denote an input graph, where $X \in \mathbb{R}^{N \times D}$ contains the D-dimensional representations of N nodes and $A \in \mathbb{R}^{N \times N}$ is the adjacency matrix. GDPool uses the layer-normalized node representations as queries and keys, while retaining the original representations as values. GDPool computes the attention logits R from the queries and keys, obtaining the attention map H through row-wise softmax:

$$H_{ij} = \frac{\exp(R_{ij})}{\sum_{k=1}^{N} \exp(R_{ik})}. \tag{1}$$

GDPool defines the context-aware importance of each node as the total attention the node receives from all query nodes. Important nodes are thus identified from contextual interactions across the

entire graph rather than from independent node-wise scores. Nodes with importance above the mean $\bar{p}$ form the anchor set $\mathcal{A}$:

$$p_j = \sum_i H_{ij}, \quad \mathcal{A} = \{j \mid p_j > \bar{p}\}. \tag{2}$$

Since H is row-stochastic, $\sum_j p_j = N$ and hence $\bar{p} = 1$; the anchor criterion thus selects nodes with $p_j > 1$ without a threshold hyperparameter. GDPool adds no learnable parameters beyond LayerNorm, since the queries, keys, and values are the node representations themselves. Because node importance depends on contextual interactions within each graph, the number of anchors varies across graphs, yielding graph-specific pooling granularity. Unlike prior adaptive pooling methods that require a predefined pooling ratio or an upper bound on the number of clusters, GDPool determines the anchor set directly from the graph-wide attention distribution. A node is selected exactly when it receives more than the uniform share of attention, so the granularity is induced by the importance distribution of each graph rather than fixed in advance or estimated as a separate budget. When no node exceeds the threshold, the node with the highest importance is included in $\mathcal{A}$ to guarantee at least one anchor.

4.2 Graph-dependent assignment and coarsening

Based on the selected graph-specific anchors, GDPool constructs a hierarchical representation through soft-assignment of each node to the anchors. Specifically, the assignment score $R^{\mathcal{A}}_{ik}$ between the i-th node and the k-th anchor $a_k \in \mathcal{A}$ is given by the attention score $R_{a_k i}$. Normalizing the assignment scores over the C selected anchors yields the soft-assignment matrix $S \in \mathbb{R}^{N \times C}$:

$$S_{ik} = \frac{\exp(R^{\mathcal{A}}_{ik})}{\sum_{l=1}^{C} \exp(R^{\mathcal{A}}_{il})}. \tag{3}$$

Each node therefore holds continuous assignment weights over all selected anchors rather than belonging to a single anchor. The weight S_{ik} indicates how strongly anchor a_k represents node i. Using S, GDPool coarsens the input graph into a coarse graph $G' = (X', A')$ with one coarse node per anchor. The coarse node representations and adjacency matrix are given by $X' = S^\top X \in \mathbb{R}^{C \times D}$ and $A' = S^\top A S \in \mathbb{R}^{C \times C}$. Each coarse node aggregates the representations of the original nodes according to the assignment weights, so that, unlike hard clustering, an original node can contribute to multiple coarse nodes. The detailed procedure of GDPool is provided in Appendix A.

4.3 Hierarchical architecture

As shown in Figure 2, GDPool is integrated into a hierarchical architecture that stacks GDformer blocks on a graph encoder, each block consisting of a Graph Feed-Forward Network (GFFN) followed by GDPool. Given an input graph $G^{(l)}$, the GFFN updates the node representations of $G^{(l)}$, after which GDPool constructs the coarse graph $G^{(l+1)}$ as the input to the next level. Repeated application of GDformer blocks yields hierarchical graph representations with graph-dependent pooling granularity. Because graphs within a mini-batch differ in size, the graphs at each level are padded to a common size, with a mask excluding padded nodes from attention, assignment, and readout. Masked readouts from all levels are aggregated for graph-level prediction. Appendix B.1 illustrates the GDPool layer.

To preserve the structural information of the original graph at each level, we adopt the auxiliary link prediction objective $\mathcal{L}_{\text{Link}}$ and the entropy regularization $\mathcal{L}_{\text{Ent}}$ of DiffPool. Under layer-wise supervision, existing pooling methods learn structural consistency between the input and pooled graphs using the assignment matrix of the current level. In hierarchical pooling, however, the original nodes are connected to the coarse nodes at higher levels through a chain of assignment matrices. Hierarchical supervision therefore defines the objective at each level on the assignment accumulated along the hierarchy rather than on the assignment of the corresponding level alone.

Based on the accumulated assignment, we apply link prediction and entropy regularization at each level so that the hierarchical representations maintain the structural relationships of the original graph. The overall hierarchical supervision is therefore

$$\mathcal{L}_{Hier} = \sum_{l=1}^{L} \left(\mathcal{L}_{\text{Link}}(S^{(l)}_{\text{Hier}}) + \lambda \mathcal{L}_{\text{Ent}}(S^{(l)}_{\text{Hier}}) \right). \tag{4}$$

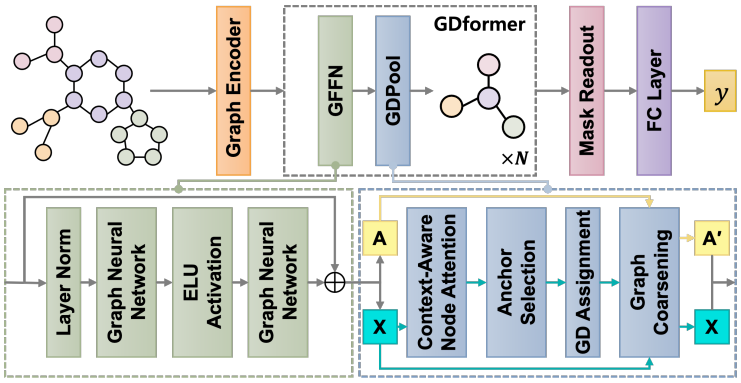


Figure 2: Overview of the proposed hierarchical graph learning architecture, which progressively learns hierarchical graph representations using GDPool.

The accumulated assignment from the first to the l-th level is $S^{(l)}_{\text{Hier}} = S^{(1)}S^{(2)}\cdots S^{(l)}$, with λ weighting the entropy regularization. Each level is thus supervised against the original graph rather than only its own coarsened graph, which provides a common reference across levels in contrast to the layer-wise objectives of Section 3. Since $S^{(l)}_{\text{Hier}}$ is row-stochastic like $S^{(l)}$, the objectives are applied without modification, and the first-level term coincides with layer-wise supervision, so the two schemes differ only in the terms above the first level. The final training objective is the sum of the prediction loss and the hierarchical supervision. A detailed complexity comparison between GDPool and existing pooling methods is provided in Appendix C.

5 Experiments

We conduct a comprehensive evaluation to assess the effectiveness of GDPool in graph property prediction and hierarchical graph representation learning. For a fair comparison, all pooling methods within each dataset share the same encoder, GFFN, and training settings, so that only the pooling operator differs. On the MOF benchmarks, graphs constructed from 3D crystal structures are encoded by MMAGNN (Teng et al., 2025), with SAGEConv (Hamilton et al., 2017) used in each GFFN. On the molecular benchmarks, the graph encoder consists of four GINEConv (Hu et al., 2020) layers to exploit the discrete bond-type edge features provided by the datasets, with GINEConv also used in each GFFN. We set the GFFN expansion ratio to 2 and the entropy weight λ to 0.05, whose effects are further analyzed in Section 5.5. The details of the architectures used in each experiment and the hyperparameter selection procedure are described in Appendix B.2.

Datasets. We evaluate GDPool on four graph regression datasets spanning MOFs and molecular graphs: ARC-MOF (Burner et al., 2023), hMOF (Wilmer et al., 2012), ZINC (Dwivedi et al., 2023), and QM9 (Ramakrishnan et al., 2014). ARC-MOF and hMOF contain relatively large material graphs and are used for gas adsorption prediction, while ZINC and QM9 contain smaller molecular graphs and are used for solubility and dipole moment prediction, respectively. Dataset statistics are summarized in Table 4. All targets are used in their native units without normalization, and both molecular benchmarks are treated as 2D graphs without atomic coordinates; reported errors are therefore absolute and comparable only across pooling operators, not to geometry-based models (details in Appendix D).

Baseline methods. We compare GDPool with hierarchical pooling methods from the three categories in Section 2: **clustering-based** (DiffPool, MinCutPool, MemPool, and BN-Pool), **node selection-based** (TopKPool, SAGPool, ASAPool, and MaxCutPool), and **edge contraction-based** (EdgePool) pooling. Except for BN-Pool, which infers the number of active clusters within a specified maximum, clustering-based methods require a predefined number of clusters. Node selection-based methods mainly coarsen graphs according to a predefined pooling ratio.

Table 1: MAE of graph pooling methods on the benchmark datasets (mean ± std over three seeds; ±0.000 denotes std below 0.0005; units in Appendix D). The best results are shown in bold.

Method	ARC-MOF	hMOF		ZINC	QM9
	CO_2@0.15	CO_2@0.1	CH_4@0.9	Solubility	Dipole Moment
DiffPool	0.184±0.014	0.159±0.002	0.137±0.010	0.323±0.007	0.483±0.012
MinCutPool	0.158±0.001	0.129±0.001	0.110±0.002	0.319±0.002	0.452±0.003
MemPool	0.157±0.000	0.129±0.000	0.110±0.000	0.316±0.002	0.448±0.002
BN-Pool	0.160±0.004	0.130±0.001	0.113±0.001	0.322±0.003	0.460±0.000
TopKPool	0.175±0.004	0.137±0.002	0.117±0.002	0.470±0.039	0.469±0.015
SAGPool	0.167±0.002	0.137±0.001	0.121±0.002	0.352±0.084	0.487±0.013
ASAPool	0.406±0.416	0.422±0.000	0.477±0.000	0.741±0.064	0.502±0.000
MaxCutPool	0.382±0.177	0.347±0.016	0.272±0.121	0.585±0.113	0.668±0.090
EdgePool	0.161±0.001	0.131±0.000	0.113±0.002	0.320±0.002	0.452±0.002
GDPool	**0.151**±0.000	**0.125**±0.002	**0.106**±0.001	**0.240**±0.015	**0.402**±0.003

5.1 Comparison with existing pooling methods

To validate the performance of GDPool, we use the four benchmark datasets. The targets are the CO_2 uptake at 0.15 bar on ARC-MOF, the CO_2 uptake at 0.1 bar and the CH_4 uptake at 0.9 bar on hMOF, the constrained solubility on ZINC, and the dipole moment on QM9. Predictive performance is evaluated with Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R^2, where lower MAE and RMSE and higher R^2 indicate better prediction. As shown in Table 1, GDPool achieves the best MAE on every prediction task across the datasets compared. GDPool therefore improves consistently on MOF and molecular graphs with different characteristics, rather than on a particular dataset or property. These results indicate that the anchor-based pooling of GDPool is effective for graph representation learning across graph types. The same trend holds for RMSE and R^2 (Tables 5 and 6). ASAPool and MaxCutPool did not train stably in our setting; we report their results as obtained and discuss this in Appendix E.

5.2 Hierarchical clustering behavior

To analyze hierarchical graph representations, we evaluate clustering behavior according to the pooling process. We compare GDPool with DiffPool, MinCutPool, MemPool, and BN-Pool, which directly use soft assignment. We measure the structural cohesion of the assignment produced at each pooling level with modularity. As shown in Table 2, the modularity of the baselines collapses once pooling levels are stacked, whereas GDPool maintains high modularity at every level. A near-uniform soft assignment gives $Q \approx 0$ by construction, so these values indicate assignments that do not follow the graph structure; such degenerate assignments have been noted previously (Bianchi et al., 2020). GDPool therefore aggregates nodes according to the structural relationships of the input graph rather than merely reducing the number of nodes. We further compare the nominal number of clusters with K_{eff}, the number of clusters actually used during pooling. Natural hierarchical coarsening activates fewer clusters at higher levels as pooling proceeds. The baselines show no clear reduction: K_{eff} at the first level is smaller than at later levels, or most clusters remain active at every level, leaving little difference from the nominal number. Stacking pooling levels therefore does not necessarily reduce the granularity of the representations. In contrast, K_{eff} of GDPool decreases with depth, using more clusters at early levels to represent fine-grained structure while activating fewer clusters at later levels to form coarse-grained representations. GDPool thus forms an appropriate pooling granularity by selecting anchors from the node representations of each graph instead of enforcing a fixed number of clusters. Both metrics are defined in Appendix F.

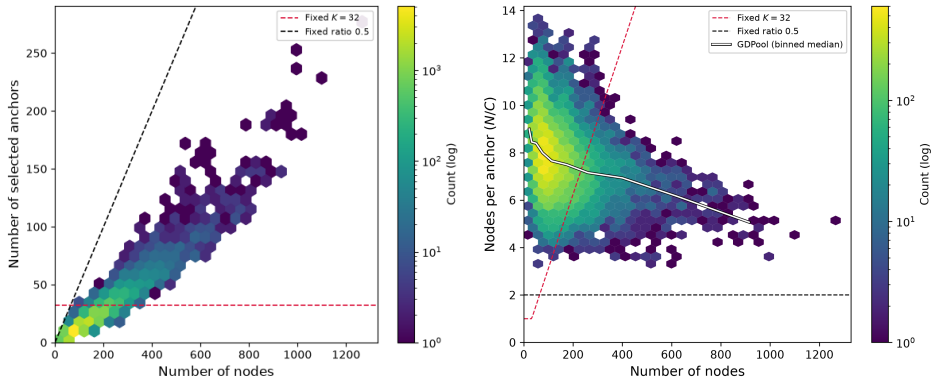
5.3 Graph-dependent anchor selection

To test whether pooling granularity adapts to each graph, we examine the number of anchors C selected by GDPool at the first pooling level as a function of the number of nodes N. As shown in

Table 2: Layer-wise Modularity and K_{eff} for CH_4@0.9 on the hMOF dataset.

Method	**Modularity**			$\boldsymbol{K}_{\text{eff}}$		
	Layer 1	Layer 2	Layer 3	Layer 1	Layer 2	Layer 3
DiffPool	$0.00_{\pm 0.01}$	$0.00_{\pm 0.01}$	$0.00_{\pm 0.02}$	$6.16_{\pm 1.07}$	$2.21_{\pm 0.45}$	$2.76_{\pm 0.53}$
MinCutPool	$0.10_{\pm 0.08}$	$0.02_{\pm 0.03}$	$0.01_{\pm 0.01}$	$3.67_{\pm 2.42}$	$9.69_{\pm 4.59}$	$7.01_{\pm 0.94}$
MemPool	$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$32.00_{\pm 0.00}$	$16.00_{\pm 0.00}$	$8.00_{\pm 0.00}$
BN-Pool	$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$31.88_{\pm 0.01}$	$15.79_{\pm 0.00}$	$7.85_{\pm 0.00}$
GDPool	$0.39_{\pm 0.01}$	$0.38_{\pm 0.00}$	$0.21_{\pm 0.00}$	$17.35_{\pm 1.03}$	$9.53_{\pm 0.34}$	$5.43_{\pm 0.09}$

Figure 3a, C increases roughly in proportion to N, following neither a fixed number of clusters nor the default pooling ratio of 0.5 used by ratio-based methods. Around the trend, graphs of similar size receive different numbers of anchors, so the granularity reflects the structure of each graph. Figure 3b shows the resulting cluster size. The median number of nodes per anchor stays between 5 and 9, whereas a fixed $K = 32$ would let it reach about 40. The number of anchors therefore scales with graph size to some extent, while graphs with the same number of nodes still receive different numbers of anchors depending on their structure. For further details on adaptive granularity, see Appendix G.1. To verify that the variation does not stem from training randomness, we analyze models trained with three random seeds on the same graphs. The ICC of the number of selected anchors is 0.971, indicating that GDPool selects a consistent number of anchors for the same graph. As shown in Table 7, the Jaccard similarity between anchor sets from different seeds is 0.581, far above the 0.071 obtained for random sets of the same size. The differences across graphs are therefore reproducible across seeds, reflecting the structure of each graph rather than the initialization (details in Appendix G.2). We replace the graph-dependent anchor selection of GDPool by fixed top-K and



(a) (b)

Figure 3: Anchor selection by GDPool at the first pooling level for CH_4@0.9 on the hMOF dataset. (a) Number of anchors. (b) Number of nodes per anchor. Red and black dashed lines indicate a fixed number of clusters and a fixed pooling ratio; color indicates graphs per bin on a logarithmic scale.

fixed-ratio rules while keeping the rest of the model unchanged. Table 8 in Appendix G.3 shows that fixed rules at their default settings waste anchors or fail to coarsen the hierarchy; a ratio matched to the anchor counts of GDPool reaches comparable accuracy, yet still leaves 29% of its first-level anchors unused, whereas GDPool activates 89%. GDPool thus reaches this budget without tuning the number of clusters or the ratio, and its high activation rate is a prerequisite for the cluster-level analysis of the resulting hierarchy. We conduct a case study on HKUST-1 (Chui et al., 1999), a representative MOF for CO_2 adsorption. In its crystal structure, all 24 Cu paddlewheels and at least 90 of the 96 carboxylate-bearing ring carbons are represented by anchors in every training seed. These anchors coincide with the two kinds of vertices of the underlying net, the Cu paddlewheel nodes that host the primary CO_2 binding sites and the branch points of the linkers (Appendix G.4).

5.4 GRADIENT CONFLICT DIAGNOSIS

Section 3 analyzed how objectives at different pooling levels can produce optimization conflicts in hierarchical graph learning. We measure the gradient similarity of GDPool during training to analyze the effect of hierarchical supervision on gradient interactions. As shown in Figure 1b, GDPool with hierarchical supervision maintains an alignment of approximately 0.6 to 0.8 throughout training, whereas conventional pooling methods with layer-wise supervision remain below 0.1 or become intermittently negative. We further compare the same GDPool architecture under hierarchical and layer-wise supervision. Figure 6a in Appendix H.1 shows the alignment over training epochs, where hierarchical supervision keeps the alignment both high and stable. Alignment depends not only on the supervision scheme but also on the anchor selection rule. Under the same hierarchical supervision, the default fixed rules lose gradient alignment and the matched ratio decays throughout training, whereas GDPool remains at about 0.6 in Figure 6b. Hierarchical supervision and graph-dependent anchor selection thus mitigate the conflict from layer-wise supervision and ill-fitting budgets, respectively. The advantage of GDPool does not hinge on the supervision scheme: supervising only the final cumulative assignment matches hierarchical supervision in accuracy (MAE 0.152 vs. 0.151), whereas hierarchical supervision yields a markedly more compact and coherent hierarchy. Layer-wise supervision degrades both accuracy and structure (Appendix H.2). Under identical hierarchical supervision, GDPool achieves lower MAE than DiffPool at either λ on hMOF (Appendix H.3).

5.5 SENSITIVITY ANALYSIS

We analyze the sensitivity of GDPool to the entropy loss weight λ, the number of hierarchical pooling levels, and the GFFN expansion ratio. As shown in Table 12, MAE is insensitive to λ, whereas modularity increases overall as λ grows. K_{eff} shows no clear trend but rises slightly overall. We adopt $\lambda = 0.05$, which improves modularity without loss of predictive performance. To analyze the effect of pooling depth, we compare models trained with one and two pooling levels. As shown in Table 13, GDPool achieves lower MAE than existing pooling methods in both settings, together with the best modularity at the first pooling level. The initial clustering behavior of GDPool therefore remains stable under changes in pooling depth. In Table 14, varying the GFFN expansion ratio makes little difference in performance. The analyses cover a GINEConv-based model and a SAGEConv-based model, with GDPool maintaining stable performance under hyperparameter and architectural changes in both cases.

6 CONCLUSION

We proposed GDPool, which adaptively determines pooling granularity according to the structural characteristics of the input graph. GDPool selects a different number of anchors for each graph using the contextual importance computed by self-attention, constructing a coarse graph through soft-assignment of each node to the selected anchors. Hierarchical graph representations are thereby learned without fixing the number of clusters or the pooling ratio in advance. Our gradient alignment analysis reveals that auxiliary objectives at different pooling levels can send conflicting gradients to shared parameters. Supervising each level only through a local objective, as in existing practice, therefore does not guarantee coherent learning of the hierarchy.

On the four benchmarks, ARC-MOF, hMOF, ZINC, and QM9, GDPool consistently outperformed all compared pooling methods. In the layer-wise analysis of modularity and K_{eff}, existing methods either retain the nominal number of clusters at every level or vary K_{eff} non-monotonically across levels, whereas K_{eff} of GDPool decreases monotonically with depth while maintaining high modularity. GDPool thus learns a granularity suited to the structure of the input graph rather than merely reducing the number of nodes.

Our study has several limitations. The number of anchors is determined by the mean attention score, so the selection criterion itself is not learned, and hierarchical supervision reuses existing clustering objectives rather than ones designed for cumulative assignments. Future work could explore learned granularity criteria and conflict-aware optimization, and evaluate GDPool on larger and more diverse graph-level tasks. Given the stable number of nodes per anchor across graph sizes and the high activation rates of the resulting clusters, GDPool can also be extended to assign predicted properties to cluster-level substructures.

AI USE STATEMENT

In this work, we have not used generative AI tools to generate experimental results, or make final scientific claims, and other required disclosure tasks are not applicable to this work. Additionally, we used generative AI tools for language editing and proofreading. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The detailed pseudocode of GDPool is provided in Appendix A. Details necessary to reproduce our experiments, including dataset descriptions, model configurations, training settings, and evaluation metrics, are provided in Section 5, Appendix B, Appendix D and Appendix F. Additional experimental results, including gradient alignment and sensitivity analyses, are reported in the Appendix. An anonymized implementation and configuration files are provided as supplementary materials.

REFERENCES

Carlo Abate and Filippo Maria Bianchi. Maxcutpool: differentiable feature-aware maxcut for pooling in graph neural networks. In *International Conference on Learning Representations*, volume 2025, pp. 52506–52533, 2025.

Davide Bacciu, Alessio Conte, and Francesco Landolfi. Generalizing downsampling from regular data to graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 6718–6727, 2023.

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

Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. Spectral clustering with graph neural networks for graph pooling. In *International conference on machine learning*, pp. 874–883. PMLR, 2020.

Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008 (10):P10008, 2008.

N Scott Bobbitt, Kaihang Shi, Benjamin J Bucior, Haoyuan Chen, Nathaniel Tracy-Amoroso, Zhao Li, Yangzesheng Sun, Julia H Merlin, J Ilja Siepmann, Daniel W Siderius, et al. Mofx-db: An online database of computational adsorption data for nanoporous materials. *Journal of Chemical & Engineering Data*, 68(2):483–498, 2023.

Jake Burner, Jun Luo, Andrew White, Adam Mirmiran, Ohmin Kwon, Peter G Boyd, Stephen Maley, Marco Gibaldi, Scott Simrod, Victoria Ogden, et al. Arc–mof: a diverse database of metal-organic frameworks with dft-derived partial atomic charges and descriptors for machine learning. *Chemistry of Materials*, 35(3):900–916, 2023.

Chen Cai, Dingkang Wang, and Yusu Wang. Graph coarsening with neural networks. *arXiv preprint arXiv:2102.01350*, 2021.

Cătălina Cangea, Petar Veličković, Nikola Jovanović, Thomas Kipf, and Pietro Liò. Towards sparse hierarchical graph classifiers. *arXiv preprint arXiv:1811.01287*, 2018.

Daniele Castellana and Filippo Maria Bianchi. BN-pool: Bayesian nonparametric pooling for graphs. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=3B3Zr2xfkf.

Chi Chen, Weike Ye, Yunxing Zuo, Chen Zheng, and Shyue Ping Ong. Graph networks as a universal machine learning framework for molecules and crystals. *Chemistry of Materials*, 31(9):3564–3572, 2019.

Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International conference on machine learning*, pp. 794–803. PMLR, 2018.

Stephen S-Y Chui, Samuel M-F Lo, Jonathan PH Charmant, A Guy Orpen, and Ian D Williams. A chemically functionalizable nanoporous material [cu3 (tma) 2 (h2o) 3] n. *science*, 283(5405): 1148–1150, 1999.

Yongchul G Chung, Emmanuel Haldoupis, Benjamin J Bucior, Maciej Haranczyk, Seulchan Lee, Hongda Zhang, Konstantinos D Vogiatzis, Marija Milisavljevic, Sanliang Ling, Jeffrey S Camp, et al. Advances, updates, and analytics for the computation-ready, experimental metal–organic framework database: Core mof 2019. *Journal of Chemical & Engineering Data*, 64(12):5985–5998, 2019.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 2016.

Frederik Diehl. Edge contraction pooling for graph neural networks. *arXiv preprint arXiv:1905.10990*, 2019.

Yunshu Du, Wojciech M Czarnecki, Siddhant M Jayakumar, Mehrdad Farajtabar, Razvan Pascanu, and Balaji Lakshminarayanan. Adapting auxiliary losses using gradient similarity. *arXiv preprint arXiv:1812.02224*, 2018.

Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020.

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.

Hongyang Gao and Shuiwang Ji. Graph u-nets. In *international conference on machine learning*, pp. 2083–2092. PMLR, 2019.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pp. 1263–1272. JMLR.org, 2017.

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

William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 1024–1034, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/5dd9db5e033da9c6fb5ba83c7a7ebea9-Abstract.html.

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. URL https://openreview.net/forum?id=HJlWWJSFDH.

Zengfeng Huang, Shengzhong Zhang, Chong Xi, Tang Liu, and Min Zhou. Scaling up graph neural networks via graph coarsening. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, pp. 675–684, 2021.

Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7482–7491, 2018.

Amir Hosein Khasahmadi, Kaveh Hassani, Parsa Moradi, Leo Lee, and Quaid Morris. Memory-based graph networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=r1laNeBYPB.

Boris Knyazev, Graham W Taylor, and Mohamed Amer. Understanding attention and generalization in graph neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/4c5bcfec8584af0d967f1ab10179ca4b-Paper.pdf.

Sung Moon Ko, Sungjun Cho, Dae-Woong Jeong, Sehui Han, Moontae Lee, and Honglak Lee. Grouping matrix based graph pooling with adaptive number of clusters. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 8334–8342, 2023.

Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. *Advances in neural information processing systems*, 34:21618–21629, 2021.

Brian Kulis and Michael I Jordan. Revisiting k-means: New algorithms via bayesian nonparametrics. *arXiv preprint arXiv:1111.0352*, 2011.

Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. In *Artificial intelligence and statistics*, pp. 562–570. Pmlr, 2015.

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

Bo Liu, Xingchao Liu, Xiaojie Jin, Peter Stone, and Qiang Liu. Conflict-averse gradient descent for multi-task learning. *Advances in neural information processing systems*, 34:18878–18890, 2021.

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. *arXiv preprint arXiv:2204.07321*, 2022.

Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with warm restarts. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=Skq89Scxx.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Andreas Loukas. Graph reduction with spectral and cut guarantees. *Journal of Machine Learning Research*, 20(116):1–42, 2019.

Andreas Loukas and Pierre Vandergheynst. Spectrally approximating large graphs with smaller graphs. In *International conference on machine learning*, pp. 3237–3246. PMLR, 2018.

Mark EJ Newman. Modularity and community structure in networks. *Proceedings of the national academy of sciences*, 103(23):8577–8582, 2006.

Mark EJ Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.

Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole Von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1(1):140022, 2014.

Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. *Advances in neural information processing systems*, 35:14501–14515, 2022.

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, pp. 5470–5477, 2020.

Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. *Advances in neural information processing systems*, 31, 2018.

Patrick E Shrout and Joseph L Fleiss. Intraclass correlations: uses in assessing rater reliability. *Psychological bulletin*, 86(2):420, 1979.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

Yukun Teng, Haoyi Tan, Wei Huang, and Guangcun Shan. Atomic-level interpretable multimodal graph neural network for predicting carbon dioxide adsorption in metal-organic frameworks. *Communications Physics*, 8(1):491, 2025.

Vincent A Traag, Ludo Waltman, and Nees Jan Van Eck. From louvain to leiden: guaranteeing well-connected communities. *Scientific reports*, 9(1):5233, 2019.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

Christopher E Wilmer, Michael Leaf, Chang Yeon Lee, Omar K Farha, Brad G Hauser, Joseph T Hupp, and Randall Q Snurr. Large-scale screening of hypothetical metal–organic frameworks. *Nature chemistry*, 4(2):83–89, 2012.

Junran Wu, Xueyuan Chen, Ke Xu, and Shangzhe Li. Structural entropy guided graph hierarchical pooling. In *International conference on machine learning*, pp. 24017–24030. PMLR, 2022.

Saining Xie and Zhuowen Tu. Holistically-nested edge detection. In *Proceedings of the IEEE international conference on computer vision*, pp. 1395–1403, 2015.

Tian Xie and Jeffrey C Grossman. Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties. *Physical review letters*, 120(14):145301, 2018.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In *International conference on machine learning*, pp. 10524–10533. PMLR, 2020.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ryGs6iA5Km.

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? *Advances in neural information processing systems*, 34:28877–28888, 2021.

Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems*, 31, 2018.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *Advances in neural information processing systems*, 33: 5824–5836, 2020.

Zhen Zhang, Jiajun Bu, Martin Ester, Jianfeng Zhang, Chengwei Yao, Zhi Yu, and Can Wang. Hierarchical graph pooling with structure learning. *arXiv preprint arXiv:1911.05954*, 2019.

Chuanxia Zheng and Andrea Vedaldi. Online clustered codebook. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 22741–22750. IEEE, 2023.

Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. Unet++: A nested u-net architecture for medical image segmentation. In *International workshop on deep learning in medical image analysis*, pp. 3–11. Springer, 2018.

A GDPool Algorithm

GDPool transforms $G = (X, A)$ into $G' = (X', A')$ based on C anchors derived from the structural characteristics of the input graph. Since C is determined dynamically for each graph, GDPool can construct hierarchical graph representations with different pooling granularities for different input graphs. The overall procedure of the GDPool is summarized in the pseudocode in Algorithm 1.

Algorithm 1 Graph-dependent anchor selection and assignment

Input: node features $X \in \mathbb{R}^{N \times D}$, valid-node mask $M \in \{0, 1\}^N$
Output: assignment matrix $S \in \mathbb{R}^{N \times C}$, anchor set $\mathcal{A} = \{a_1, \ldots, a_C\}$

$\bar{X} \leftarrow \mathrm{LayerNorm}(X)$
$Q \leftarrow \bar{X}, \quad K \leftarrow \bar{X}$
$R \leftarrow QK^\top / \sqrt{D}$
$R_{ij} \leftarrow -\infty$ if $M_i M_j = 0$
$H \leftarrow \mathrm{Softmax}_j(R)$
$p_j \leftarrow \sum_i H_{ij}$ ▷ Node importance
$N_v \leftarrow \sum_j M_j$
$\bar{p} \leftarrow \frac{1}{N_v} \sum_{j:M_j=1} p_j$ ▷ Mean node importance
$\mathcal{A} \leftarrow \{j \mid p_j > \bar{p},\ M_j = 1\}$
$a^* \leftarrow \arg\max_{j:M_j=1} p_j$
$\mathcal{A} \leftarrow \mathcal{A} \cup \{a^*\}$ ▷ Ensure at least one anchor
$R^{\mathcal{A}}_{ik} \leftarrow R_{a_k i}, \quad a_k \in \mathcal{A}$
$S_{ik} \leftarrow \mathrm{Softmax}_k\left(R^{\mathcal{A}}_{ik}\right)$ ▷ Graph-dependent soft assignment
$S_{ik} \leftarrow 0$ if $M_i = 0$
return $S, \mathcal{A}$

B Implementation Details

B.1 GDPool Layer

A schematic depiction of the GDPool layer is illustrated in Figure 4. GDPool first uses the normalized node representations as queries and keys to score node importance, and selects anchor nodes for each graph based on these scores. It then derives a soft assignment of each node to the anchors and aggregates the values of the original nodes according to this assignment to produce the coarsened graph.

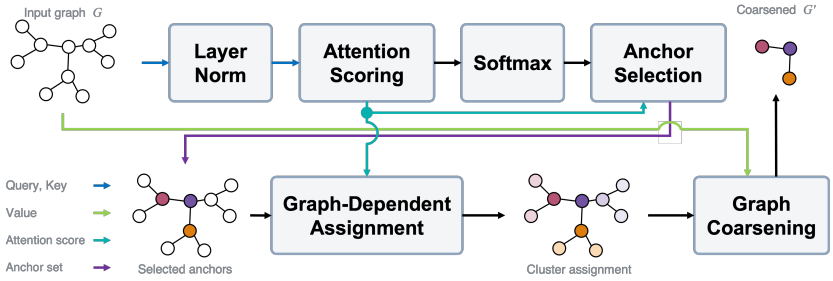


Figure 4: Scheme of the GDPool layer.

B.2 Model Configurations

The embedding dimension of the node representations was set to 64, and the batch size was fixed at 64. All models were trained with the AdamW optimizer (Loshchilov & Hutter, 2019) with a weight decay of 10^{-4}. The learning rate followed a cosine annealing schedule (Loshchilov & Hutter, 2017),

with the maximum and minimum learning rates set to 10^{-3} and 10^{-5}, respectively. On the ARC-MOF and hMOF datasets, all models were trained for 100 epochs with three hierarchical pooling levels; for pooling methods that require the number of clusters to be specified in advance, the number of clusters at each level was set to 32, 16, and 8, respectively. On the ZINC and QM9 datasets, all models were trained for 300 epochs with two hierarchical pooling levels, and the number of clusters at each level was set to 12 and 6, respectively, for such methods. For the single-level models in Table 13, the number of clusters was set to 12. For pooling-ratio-based methods, the default pooling ratio of each method was used on all datasets. All experiments were repeated three times with different random seeds, and the same three seeds were used for every method. For each run, the checkpoint with the lowest validation MAE is used for evaluation on the test set. Finally, all experiments were conducted on a single NVIDIA RTX PRO 6000 GPU.

C Complexity and computational cost

Table 3 reports the training time per epoch and the peak GPU memory measured on the hMOF dataset. In terms of training time, GDPool takes 236.21 seconds per epoch, which is faster than MemPool, comparable to MaxCutPool, and about $1.8\times$ that of SAGPool, the fastest method. It is also more than $8\times$ faster than EdgePool, which performs iterative edge contraction. In contrast, its peak memory usage of 10.56 GB is higher than that of all other methods. This is because GDPool must retain both the attention logits and the softmax outputs, and additionally constructs the assignment to the anchors on top of them. Moreover, since the graphs in a mini-batch are padded to $C_{\max} = \max_b C_b$, where C_b is the number of anchors in the b-th graph, a single graph with many anchors forces the memory of the entire batch to be allocated accordingly. As shown in Figure 3, the number of anchors scales with graph size and the size distribution of hMOF graphs is right-skewed. The distribution of anchor counts has a long right tail, so the actual memory usage is governed by the maximum within a batch rather than by the average number of anchors. The relatively large standard deviation of the training time is likewise attributable to the variation of $C_{\max}$ across batches. Since the number of anchors is determined during the forward pass, batches cannot be formed by exact anchor count at the data loading stage. Because the number of anchors scales with graph size, however, bucketing graphs by size would largely reduce this overhead; we leave this to future work.

Table 3: Memory complexity and measured cost on the hMOF dataset. N and E denote the number of nodes and edges. Time is the duration of one training epoch. Peak memory is the maximum GPU memory observed during that epoch.

Method	Memory	Time (sec)	Peak mem. (GB)
DiffPool	$O(N^2)$	$160.72_{\pm 2.43}$	3.90
MinCutPool	$O(N^2)$	$188.56_{\pm 3.16}$	3.73
MemPool	$O(N^2)$	$293.36_{\pm 2.83}$	3.68
BN-Pool	$O(N^2)$	$153.07_{\pm 0.87}$	4.35
TopKPool	$O(N+E)$	$147.15_{\pm 1.03}$	2.89
SAGPool	$O(N+E)$	$128.98_{\pm 0.77}$	2.80
ASAPool	$O(N+E)$	$162.76_{\pm 1.41}$	4.01
MaxCutPool	$O(N+E)$	$213.71_{\pm 15.55}$	2.98
EdgePool	$O(N+E)$	$1939.43_{\pm 332.43}$	2.82
GDPool	$O(N^2)$	$236.21_{\pm 21.80}$	10.56

D Datasets details

The statistics of the benchmark datasets used in this paper, ARC-MOF, hMOF, ZINC, and QM9, are reported in Table 4. For ARC-MOF and hMOF, the labels are gas uptake capacities obtained from GCMC simulations at the pressures indicated in each column header, reported in mol/kg. The hMOF isotherms were obtained from MOFX-DB (Bobbitt et al., 2023), which hosts simulated isotherms

for the hypothetical MOF structures of Wilmer et al. (2012). For ZINC we use the 12,000 graph subset, whose label is the constrained solubility $\mathrm{logP} - \mathrm{SAS} - \mathrm{cycles}$. For QM9, we predict the dipole moment μ, reported in Debye. We use all hMOF structures with CO_2 adsorption data in MOFX-DB (137,652 structures) and, for ARC-MOF, the ARCMOF_20241004 release with all structures that have post-combustion CO_2 adsorption data (post_comb_vsa-CO2.csv; 242,390 structures), without further filtering. The hMOF dataset is randomly split into training, validation, and test sets with a 70%/15%/15% ratio. The ARC-MOF dataset is split with the same ratio by a coverage-aware random procedure: for each element, from rarest to most common, one unassigned structure containing it is placed in each split so that every element appears in all three splits, and the remaining structures are assigned at random. QM9 is randomly split with an 80%/10%/10% ratio, and for ZINC we use the default split provided by PyTorch Geometric. The same splits are used for all compared methods. ARC-MOF contains several variants of the same parent structure (e.g., different solvent-removal treatments); because a single split is shared by all methods, the methods are compared under identical conditions, but absolute errors may be optimistic. The 3D atomic coordinates available in QM9 are not used. All targets are used in their native units, without further normalization or rescaling. Reported errors are therefore directly interpretable in physical units, but are not comparable to task-specialized models that are tuned per dataset and employ target normalization.

Table 4: Statistics of datasets.

Dataset	**# Graphs**	**Avg. # Nodes**	**Avg. # Edges**
ARC-MOF	242,390	157.57	1890.8
hMOF	137,652	141.39	1696.7
ZINC	12,000	23.2	49.8
QM9	130,831	18.0	37.3

E Additional results for property prediction

The main paper reports MAE results for property prediction, while RMSE and R^2 are also used as evaluation metrics. The corresponding RMSE and R^2 results are provided in Tables 5 and 6, respectively, for completeness. ASAPool and MaxCutPool did not train stably in our setting. ASAPool converges to a near-constant predictor on both hMOF targets, yielding negative R^2 in Table 6, and MaxCutPool shows the largest variance among all compared methods. Both were given the same backbone, training budget, and hyperparameter search as every other baseline, and we report their results as obtained rather than excluding them. Two factors may contribute. First, their node-selection mechanisms may be mismatched to the large, dense material graphs used here. Second, both are typically evaluated with a single pooling layer, whereas our setting stacks two or three; consistently, Table 13 shows that all four node-selection baselines degrade as pooling depth increases from one to two levels, with MaxCutPool degrading the most, whereas GDPool improves. These observations concern the behavior of these methods under deep hierarchical stacking and should not be read as a general assessment of them.

F Evaluation metrics of hierarchical clustering behavior

Modularity is computed directly from the soft assignment as

$$Q = \frac{1}{2m}\,\mathrm{tr}\big(S^\top B S\big), \qquad B = A - \frac{dd^\top}{2m}, \tag{5}$$

where B is the modularity matrix (Newman, 2006), d is the degree vector, and m is the number of edges. For hard assignments this reduces to the standard modularity, and near-uniform soft assignments yield $Q \approx 0$. At level $l > 1$, A denotes the coarse adjacency produced by the preceding level, so modularity values at deeper levels are computed on progressively smaller graphs and are not directly comparable across methods that coarsen at different rates.

The effective number of clusters K_{eff} measures how many clusters are actually used (Zheng & Vedaldi, 2023). For each graph, the total assignment mass received by each cluster is normalized

Table 5: RMSE of graph pooling methods on the benchmark datasets.

Method	ARC-MOF	hMOF		ZINC	QM9
	CO_2@0.15	CO_2@0.1	CH_4@0.9	Solubility	Dipole Moment
DiffPool	$0.356_{\pm 0.031}$	$0.316_{\pm 0.002}$	$0.247_{\pm 0.019}$	$0.897_{\pm 0.004}$	$0.696_{\pm 0.013}$
MinCutPool	$0.295_{\pm 0.003}$	$0.262_{\pm 0.003}$	$0.200_{\pm 0.003}$	$0.811_{\pm 0.010}$	$0.659_{\pm 0.004}$
MemPool	$0.293_{\pm 0.001}$	$0.262_{\pm 0.001}$	$0.202_{\pm 0.001}$	$0.826_{\pm 0.004}$	$0.654_{\pm 0.004}$
BN-Pool	$0.297_{\pm 0.011}$	$0.263_{\pm 0.002}$	$0.205_{\pm 0.002}$	$0.824_{\pm 0.011}$	$0.665_{\pm 0.003}$
TopKPool	$0.325_{\pm 0.005}$	$0.275_{\pm 0.003}$	$0.213_{\pm 0.002}$	$0.863_{\pm 0.040}$	$0.682_{\pm 0.016}$
SAGPool	$0.309_{\pm 0.002}$	$0.273_{\pm 0.001}$	$0.216_{\pm 0.004}$	$0.941_{\pm 0.029}$	$0.698_{\pm 0.008}$
ASAPool	$0.670_{\pm 0.621}$	$0.733_{\pm 0.000}$	$0.732_{\pm 0.000}$	$1.003_{\pm 0.072}$	$0.709_{\pm 0.002}$
MaxCutPool	$0.647_{\pm 0.273}$	$0.597_{\pm 0.037}$	$0.445_{\pm 0.182}$	$1.136_{\pm 0.090}$	$0.959_{\pm 0.142}$
EdgePool	$0.304_{\pm 0.001}$	$0.264_{\pm 0.002}$	$0.207_{\pm 0.003}$	$0.813_{\pm 0.005}$	$0.661_{\pm 0.002}$
GDPool	$\mathbf{0.282}_{\pm 0.003}$	$\mathbf{0.258}_{\pm 0.004}$	$\mathbf{0.198}_{\pm 0.001}$	$\mathbf{0.580}_{\pm 0.039}$	$\mathbf{0.601}_{\pm 0.004}$

Table 6: R^2. of graph pooling methods on the benchmark datasets.

Method	ARC-MOF	hMOF		ZINC	QM9
	CO_2@0.15	CO_2@0.1	CH_4@0.9	Solubility	Dipole Moment
DiffPool	$0.894_{\pm 0.018}$	$0.786_{\pm 0.003}$	$0.855_{\pm 0.022}$	$0.802_{\pm 0.002}$	$0.786_{\pm 0.008}$
MinCutPool	$0.928_{\pm 0.002}$	$0.853_{\pm 0.004}$	$0.905_{\pm 0.003}$	$0.838_{\pm 0.004}$	$0.808_{\pm 0.003}$
MemPool	$0.929_{\pm 0.001}$	$0.853_{\pm 0.002}$	$0.903_{\pm 0.001}$	$0.832_{\pm 0.002}$	$0.811_{\pm 0.002}$
BN-Pool	$0.927_{\pm 0.006}$	$0.852_{\pm 0.003}$	$0.900_{\pm 0.002}$	$0.833_{\pm 0.005}$	$0.805_{\pm 0.002}$
TopKPool	$0.912_{\pm 0.003}$	$0.838_{\pm 0.003}$	$0.893_{\pm 0.002}$	$0.817_{\pm 0.017}$	$0.794_{\pm 0.010}$
SAGPool	$0.921_{\pm 0.001}$	$0.840_{\pm 0.001}$	$0.890_{\pm 0.004}$	$0.782_{\pm 0.013}$	$0.784_{\pm 0.005}$
ASAPool	$0.414_{\pm 0.875}$	$-0.150_{\pm 0.000}$	$-0.267_{\pm 0.000}$	$0.752_{\pm 0.035}$	$0.778_{\pm 0.002}$
MaxCutPool	$0.612_{\pm 0.263}$	$0.235_{\pm 0.096}$	$0.478_{\pm 0.338}$	$0.681_{\pm 0.049}$	$0.587_{\pm 0.117}$
EdgePool	$0.924_{\pm 0.000}$	$0.851_{\pm 0.002}$	$0.899_{\pm 0.002}$	$0.838_{\pm 0.002}$	$0.807_{\pm 0.001}$
GDPool	$\mathbf{0.934}_{\pm 0.001}$	$\mathbf{0.857}_{\pm 0.004}$	$\mathbf{0.907}_{\pm 0.001}$	$\mathbf{0.917}_{\pm 0.011}$	$\mathbf{0.840}_{\pm 0.002}$

into a distribution over clusters, and K_{eff} is the perplexity of this distribution, i.e., the exponential of its entropy. It equals the number of clusters when all clusters receive equal mass and approaches one when the assignment collapses onto a single cluster. Both metrics are computed per graph and averaged over the test set.

G Anchor selection analyses

G.1 Adaptive pooling granularity

We discuss in detail the visualization of pooling granularity presented in Section 5.3. For each graph in the hMOF test set, we recorded the number of original nodes N together with the number of anchors C selected at the first pooling level, averaged over three seeds. Note that, unlike K_{eff}, C refers to the number of anchors actually selected by the anchor selection step. As shown in Figure 3a, C grows approximately linearly with N. A linear fit through the origin gives $C \approx 0.15\,N$ (Pearson $r = 0.96$), and C/N lies between 0.10 and 0.17 for 80% of the graphs (10th–90th percentiles, median 0.13). GDPool thus behaves like a ratio-based pooling method whose ratio is not set in advance but emerges from the anchor criterion $p_j > \bar{p}$, and the emergent ratio differs from both a fixed number of clusters ($K = 32$) and the default pooling ratio of 0.5. Around this trend, graphs of similar size receive different numbers of anchors. Within a 50-node size range, C varies by a factor of about 1.6 between the 10th and 90th percentiles, so the granularity reflects not only the size of each graph but also its structure. Figure 3b shows the same data as nodes per anchor, N/C. GDPool keeps this quantity within a narrow range, whereas a fixed $K = 32$ would let it grow from

1 to 40 across the graph sizes in the test set. On average, GDPool selects 19.60 anchors per graph, of which $K_{\text{eff}} = 17.35$ are active, corresponding to an activation rate of approximately 89%. In contrast, existing pooling methods either fail to form proper clusters or activate fewer than 20% of their clusters. This shows that GDPool not only allocates anchors in proportion to graph size but also actually utilizes most of the clusters it forms.

The motivation for graph-dependent granularity is interpretability. Properties of both materials and molecules are governed by local substructures, such as metal nodes and linkers in MOFs or functional groups in molecules, and we ultimately aim to attribute predicted properties to such substructures at the cluster level, which we leave to future work. This requires cluster sizes set by local structure rather than by graph size. A fixed number of clusters cannot provide this. A fixed pooling ratio keeps the cluster size constant but assigns the same budget to every graph of the same size regardless of its structure, whereas GDPool varies the budget among graphs of the same size.

G.2 Seed robustness of anchor selection

To assess the robustness of anchor selection to the choice of random seed, we measured the consistency of both the number of selected anchors and their node identities across three different seeds. To quantify the consistency of the number of selected anchors, we used the one-way random-effects ICC, i.e., ICC(1,1) in the notation of Shrout & Fleiss (1979). Treating each graph as the unit of measurement and each random seed as a repeated measurement, we compute the ICC as follows:

$$\mathrm{ICC} = \frac{\mathrm{MS}_B - \mathrm{MS}_W}{\mathrm{MS}_B + (s-1)\mathrm{MS}_W}, \tag{6}$$

where MS_B is the between-graph mean square, which captures the variation across graphs, MS_W is the within-graph mean square, which captures the variation across seeds within the same graph, and s is the number of random seeds. An ICC close to 1 therefore indicates that the variation arising from differences between graphs dominates the variation induced by the random seed, i.e., that the number of anchors is stable with respect to the seed. This experiment was conducted on the first pooling layer of the model trained on the CH_4@0.9 of the hMOF dataset. The within-graph standard deviation across seeds ($\sqrt{\mathrm{MS}_W}$) was 3.02, whereas the standard deviation of the per-graph mean anchor count was 17.54, yielding an ICC of 0.971. This demonstrates that the number of anchors selected for the same graph remains consistent even when the random seed is changed. Because C scales with N, part of this agreement reflects graph size. The ICC of the size-normalized count C/N is 0.67, so the graph-specific deviations from the size trend are also largely reproducible across seeds.

To assess the consistency of the selected nodes as well as their number, we additionally measured the Jaccard similarity between anchor sets. For each pair of seeds, we computed the Jaccard similarity per graph and averaged it over all graphs; the final score is the mean over the three seed pairs. Table 7 reports this score together with a random baseline that selects the same number of nodes uniformly at random for each graph. The anchors selected under different seeds overlap far more than expected from random selection. These results show that the anchor selection of GDPool is stable under seed changes in terms of both the number of anchors and the identity of the selected nodes.

Table 7: Jaccard similarity of anchor sets across different random seeds and the corresponding random-selection baselines for CH_4@0.9 on the hMOF dataset.

Method	**Pair 1**	**Pair 2**	**Pair 3**	**Avg.**
Random	0.071	0.074	0.069	0.071
GDPool	**0.554**	**0.586**	**0.602**	**0.581**

G.3 Effect of the anchor selection rule

Table 8 replaces the adaptive anchor rule by fixed rules while keeping the rest of the model unchanged. At their default settings, both fixed rules fail to form a coherent hierarchy. With

$K = 32/16/8$, only 14.7 of 32 first-level anchors are active and K_{eff} does not decrease from the first to the second level (14.7 to 14.8), so the first level barely coarsens the graph. With a ratio of 0.5, 70 anchors are allocated at the first level but 17 are active, K_{eff} increases from the first to the second level (17.5 to 32.3), and accuracy degrades (MAE 0.131, $p = 0.02$). A ratio matched to the anchor counts of GDPool (0.15 at the first level and 0.5 thereafter) reaches comparable accuracy (MAE 0.124 vs. 0.125, within seed variance) and a monotonically decreasing K_{eff}, but activates only 71% of its first-level anchors (15.4 of 21.5), because the same fraction is applied to every graph of a given size regardless of its structure. This setting was moreover read off the output of GDPool and would otherwise require a per-level, per-dataset search. GDPool obtains the same budget without any setting, allocates it per graph, and activates 89% of its first-level anchors (17.4 of 19.7). Using the resulting clusters in downstream analysis requires both properties. Modularity measures whether clusters follow the graph structure, and the activation rate measures whether allocated clusters are actually populated. A cluster that follows the structure but is empty cannot be analyzed, and a cluster that is populated but does not follow the structure cannot be interpreted. Since the anchor-based rules have similar modularity (0.38–0.43 at the first two levels), the activation rate is what separates them for downstream use, and GDPool is highest in this respect.

Table 8: Effect of the anchor selection rule on GDPool for CO_2@0.1 on the hMOF dataset. Default settings are $K = 32/16/8$ and a ratio of 0.5 at every level; the matched ratio uses 0.15 at the first level and 0.5 thereafter, which reproduces the mean number of anchors of GDPool. Active is the fraction of allocated anchors that are used. #Anchors, K_{eff}, and Active are reported per pooling level (L1 / L2 / L3).

Anchor rule	**MAE**	**#Anchors**	$\boldsymbol{K}_{\text{eff}}$	**Active (%)**
Fixed K	0.126±0.002	31.9 / 16.0 / 8.0	14.7 / 14.8 / 7.5	46.1 / 92.5 / 94.2
Fixed ratio	0.131±0.002	70.4 / 35.4 / 18.0	17.5 / 32.3 / 16.2	24.8 / 91.1 / 90.0
Fixed ratio (matched)	**0.124**±0.001	21.5 / 11.0 / 5.8	15.4 / 10.4 / 5.6	71.3 / 94.7 / 97.5
GDPool	0.125±0.002	19.7 / 10.0 / 5.7	17.4 / 9.6 / 5.5	**88.6 / 95.5 / 97.8**

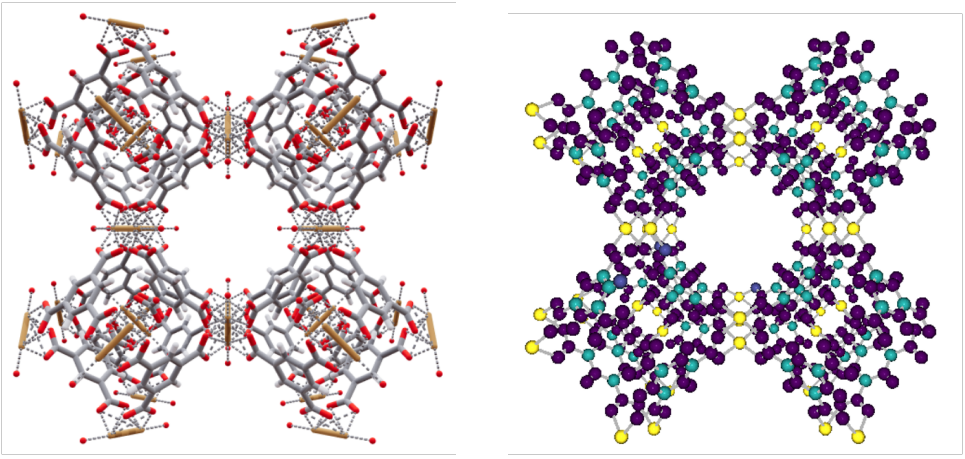
G.4 Case study on the HKUST-1

To examine where the anchor criterion places anchors in a real framework, we apply the GDPool model trained on hMOF (CO_2 at 0.1 bar) to the experimental crystal structure of HKUST-1, $Cu_3(BTC)_2$ (Chui et al., 1999). Coordinated and free water molecules are removed with the CoRE MOF cleaning procedure (Chung et al., 2019), which yields the activated framework with 624 atoms ($Cu_{48}C_{288}O_{192}H_{96}$); the structure is then converted into a graph with the same preprocessing as the hMOF dataset. Table 9 summarizes the first-level anchors for the three training seeds. For seed 42, the first pooling level selects 148 anchors: all 48 Cu atoms, 95 of the 96 carboxylate-bearing ring carbons, one symmetry class of the BTC linker, and five further atoms (one C–H ring carbon, three O, one H), so essentially every tritopic linker contributes three anchors at the positions where its arms leave the ring. The pattern holds across seeds: the 142–148 anchors are drawn from the same two symmetry classes, namely 40–48 Cu atoms, with every one of the 24 Cu paddlewheels retaining at least one Cu anchor, and 90–95 of the 96 carboxylate-bearing carbons, whereas none of the 96 carboxylate carbons is selected in any seed. The anchors therefore coincide with the two kinds of vertices of the underlying net, the Cu paddlewheel nodes and the branch points of the linkers, and the first-level clusters partition the framework into metal coordination spheres and linker arms.

Table 9: Anchors of GDPool on HKUST-1 for the three training seeds of the hMOF CO_2@0.1 model. C_{ipso}: ring carbons bearing a carboxylate; C_{H}: ring carbons bearing hydrogen; C_{COO}: carboxylate carbons. Paddlewheels: Cu_2 paddlewheels with at least one Cu anchor.

Seed	**#Anchors**	**Cu**	**Paddlewheels**	$\mathbf{C}_{\text{ipso}}$	$\mathbf{C}_{\text{H}}$	$\mathbf{C}_{\text{COO}}$	**O**	**H**
42	148 (23.7%)	48	24	95	1	0	3	1
43	142 (22.8%)	40	24	90	7	0	4	1
44	144 (23.1%)	47	24	94	0	0	3	0

Figure 5 shows the crystal structure and the nodes selected as anchors. Only atoms inside one unit cell are drawn in Figure 5a, so paddlewheels at the cell boundary appear without their ligands, whereas Figure 5b completes such fragments; the graph itself uses periodic neighbors, so the model sees complete coordination environments. Because symmetry-equivalent atoms have identical local environments, they receive nearly identical importance scores and are selected or rejected together, as for the 48 Cu atoms and 95 of the 96 carboxylate-bearing carbons in seed 42. The anchor count is thus determined in units of symmetry classes ($48 + 96$), which is why the anchor fraction of this highly symmetric structure (23–24%) exceeds the hMOF average of about 15%. When a class lies close to the threshold, small numerical differences decide, which is the origin of the partial selections in Table 9, such as 40 of the 48 Cu atoms in seed 43. A fixed budget cannot reproduce this selection: top-K with $K = 32$ must pick 32 of the 48 equivalent Cu atoms, and a fixed ratio of 0.15 (94 anchors) must split the 96-atom carbon class. This case study illustrates why the number of clusters must be determined by the graph itself rather than fixed in advance.



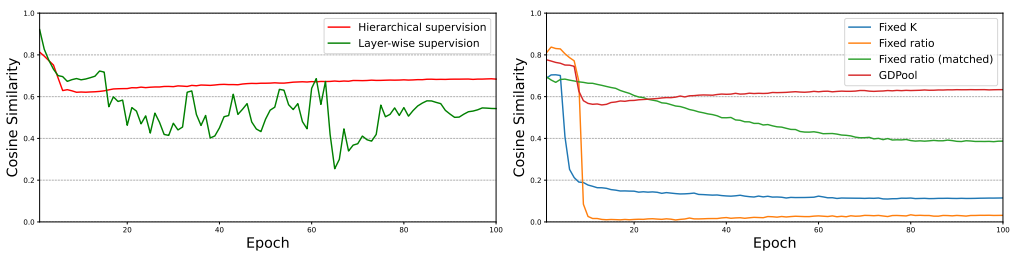
(a) (b)

Figure 5: Anchors on HKUST-1. (a) Crystal structure of the activated framework (Cu orange, O red, C gray, H white). (b) The same unit cell viewed along the c axis with first-level anchors highlighted: Cu (yellow), carbon anchors (cyan; 95 of 96 are carboxylate-bearing ring carbons), O/H (blue); non-anchor atoms are purple.

H Gradient alignment and supervision scheme

H.1 Training dynamics of gradient alignment

This section provides results for the gradient alignment presented in Section 5.4. The training dynamics are reported in Figure 6a. Final-level supervision has a single auxiliary objective, so inter-level alignment is undefined and it is omitted from Figure 6a. Under layer-wise supervision, the alignment of GDPool remains positive but lower and unstable; we relate this to the resulting hierarchy in Appendix H.2. The anchor selection rule also affects gradient alignment. Figure 6b compares the rules of Table 8 under the same hierarchical supervision. For the default fixed rules, the alignment between level objectives collapses within the first ten epochs, to about 0.1 for the fixed K and near zero for the ratio of 0.5. The matched ratio decays throughout training to about 0.4, whereas GDPool remains at about 0.6. The resulting order of the four rules coincides with the order of their first-level activation rates in Table 8. A plausible reason is that fixed rules impose the number of first-level anchors regardless of the graph, so some allocated anchors attract almost no nodes and form nearly empty clusters. The objective at the next level must nevertheless coarsen these clusters, which lowers its agreement with the first-level objective. The two measures thus point, structurally and in optimization, to the same limitation of fixed rules in forming active clusters.



(a) (b)

Figure 6: Training-time gradient alignment across pooling levels. (a) Hierarchical vs. layer-wise supervision on the ARC-MOF dataset. (b) Anchor selection rules under hierarchical supervision for CO_2@0.1 on the hMOF dataset.

H.2 Effect of the supervision scheme

Hierarchical supervision differs from layer-wise supervision only in the arguments of the objectives: at level l, the former evaluates $\mathcal{L}_{\text{Link}}$ and $\mathcal{L}_{\text{Ent}}$ on the accumulated assignment $S^{(l)}_{\text{Hier}}$ against the original adjacency $A^{(0)}$, the latter on the local assignment $S^{(l)}$ against the coarsened adjacency $A^{(l-1)}$. The two schemes share the first-level term, and the substitution above it changes the optimization in two ways. First, the reference graph becomes fixed. Under layer-wise supervision, $A^{(l-1)}$ is produced by the lower levels and moves during training, and each objective requires only that the pooled graph and the assignment agree with each other: a near-uniform assignment yields an almost constant $A^{(l)}$ that another near-uniform assignment matches up to scale, so no level is required to follow the input structure, consistent with the near-uniform assignments under layer-wise supervision in Table 10. Under hierarchical supervision, every level is compared with the same data-defined graph $A^{(0)}$. Second, the objectives become coupled: since $S^{(l)}_{\text{Hier}} = S^{(1)} \cdots S^{(l)}$, each assignment matrix receives gradients from every objective at or above its level, all measured against the same graph, which is reflected in the higher and more stable alignment in Figure 6a. Final-level supervision keeps only the $l = L$ term of Eq. 4. Table 10 compares the three schemes. Final-level supervision matches hierarchical supervision in accuracy while still outperforming all baselines, so the predictive advantage of GDPool stems from the operator rather than the supervision. The intermediate terms of Eq. 4 instead shape the hierarchy: without them, only the final partition must preserve the input structure, and the first level retains more anchors with lower modularity. Layer-wise supervision degrades both accuracy and structure. Under layer-wise supervision, whose alignment is positive but unstable in Figure 6a, the first-level K_{eff} of GDPool nearly equals the number of anchors (65.3 of 65.8), yet its modularity collapses to zero in Table 10. This indicates a near-uniform assignment that trivially maximizes K_{eff} rather than clusters that follow the graph structure. Across supervision schemes and anchor rules, reduced or unstable alignment thus co-occurs with a failure in one of the two properties required for downstream analysis (Appendix G.3): fixed rules leave first-level anchors inactive, whereas layer-wise supervision yields assignments that do not follow the graph structure. Only GDPool with hierarchical supervision attains both high activation and high modularity, together with high and stable alignment.

Table 10: Effect of the supervision scheme on GDPool (ARC-MOF, CO_2@0.15). #Anchors, modularity, and K_{eff} are reported per pooling level (L1 / L2 / L3).

Supervision	**MAE**	**#Anchors**	**Modularity**	$\boldsymbol{K}_{\text{eff}}$
Layer-wise	0.250±0.007	65.8 / 38.3 / 5.6	0.00 / 0.00 / 0.00	65.3 / 38.2 / 4.9
Final-level	0.152±0.001	53.7 / 24.7 / 12.1	0.16 / 0.32 / 0.35	49.5 / 23.0 / 11.5
Hierarchical (GDPool)	**0.151**±0.000	20.7 / 10.5 / 5.9	0.42 / 0.41 / 0.23	18.5 / 10.0 / 5.8

H.3 HIERARCHICAL SUPERVISION WITH A DIFFERENT ASSIGNMENT OPERATOR

We apply the same hierarchical supervision (Eq. 4) to DiffPool, whose auxiliary objectives are identical to ours, so that the comparison isolates the assignment operator. The architecture is identical to that of DiffPool in Table 1, so this variant differs from DiffPool only in the supervision and from GDPool only in the assignment operator. Other baselines use different objectives, which would confound the operator with the loss. Both methods are trained with $\lambda \in \{0.05, 0.2\}$ under the protocol of Appendix B.2 for CO_2@0.1 on the hMOF dataset. Table 11 shows Hierarchical supervision with different assignment operators. At the same λ, GDPool achieves lower MAE than DiffPool (0.125 vs. 0.130 at $\lambda = 0.05$; 0.127 vs. 0.138 at $\lambda = 0.2$), and its accuracy varies less with λ. The two operators also differ in how the allocated clusters are used. DiffPool activates only 33% and 21% of its 32 first-level clusters at $\lambda = 0.05$ and 0.2, respectively, whereas GDPool activates about 88% of its first-level anchors at both values, even though the number of anchors changes from 19.7 to 33.2. K_{eff} of GDPool decreases monotonically with depth at both values, whereas that of DiffPool barely decreases from the first to the second level at $\lambda = 0.05$ (10.6 to 9.8) and increases at $\lambda = 0.2$ (6.7 to 12.7), so its hierarchy does not coarsen progressively. DiffPool attains comparable or higher modularity, particularly at $\lambda = 0.2$, but with most of its first-level clusters inactive, these clusters cannot support cluster-level analysis (Appendix G.3). Under identical supervision, the assignment operator thus determines whether the allocated clusters are used and whether the hierarchy coarsens progressively.

Table 11: Hierarchical supervision with different assignment operators for CO_2@0.1 on the hMOF dataset. #Clusters, modularity, and K_{eff} are reported per pooling level (L1 / L2 / L3).

Method	$\boldsymbol{\lambda}$	**MAE**	**#Clusters**	**Modularity**	$\boldsymbol{K}_{\text{eff}}$
DiffPool (Hier.)	0.05	0.130±0.002	31.9 / 16.0 / 8.0	0.33 / 0.34 / 0.35	10.6 / 9.8 / 7.5
DiffPool (Hier.)	0.2	0.138±0.001	31.9 / 16.0 / 8.0	0.43 / 0.43 / 0.43	6.7 / 12.7 / 7.2
GDPool	0.05	**0.125**±0.002	19.7 / 10.0 / 5.7	0.39 / 0.38 / 0.22	17.4 / 9.6 / 5.5
GDPool	0.2	0.127±0.001	33.2 / 8.9 / 4.5	0.25 / 0.35 / 0.27	29.4 / 8.2 / 4.3

I SENSITIVITY ANALYSIS RESULTS

This section provides results for the sensitivity analysis presented in Section 5.5. The results show that GDPool maintains stable performance across a range of hyperparameter and architectural configurations, demonstrating its robustness.

Table 12: Effect of the entropy loss weight λ on the ZINC dataset.

$\boldsymbol{\lambda}$	**MAE**	**Modularity**		$\boldsymbol{K}_{\text{eff}}$	
		Layer 1	Layer 2	Layer 1	Layer 2
0.00	**0.239**±0.003	0.208±0.003	0.095±0.004	9.652±0.208	4.873±0.054
0.05	0.240±0.015	0.283±0.002	0.168±0.014	9.074±0.490	4.693±0.237
0.10	0.241±0.010	0.351±0.005	0.237±0.010	8.361±0.220	4.761±0.198
0.15	0.240±0.009	0.374±0.011	0.372±0.011	9.809±0.437	5.546±0.254
0.20	0.251±0.009	0.370±0.002	0.449±0.003	10.115±0.059	5.252±0.038

Table 13: Effect of pooling depth on performance and clustering behavior for the QM9 dataset.

Method	MAE		Modularity		K_{eff}	
	1×Pooling	2×Pooling	1×Pooling	2×Pooling	1×Pooling	2×Pooling
DiffPool	0.46±0.00	0.48±0.01	0.09±0.04	0.06±0.06	1.41±0.23	1.43±0.44
MinCutPool	0.46±0.00	0.45±0.00	0.21±0.01	0.17±0.00	6.10±0.66	6.48±0.48
MemPool	0.45±0.00	0.45±0.00	0.00±0.00	0.00±0.00	12.00±0.00	12.00±0.00
BN-Pool	0.46±0.00	0.46±0.00	0.00±0.00	0.00±0.00	11.88±0.01	11.88±0.00
TopKPool	0.45±0.00	0.47±0.01	-	-	-	-
SAGPool	0.45±0.01	0.49±0.01	-	-	-	-
ASAPool	0.47±0.00	0.50±0.00	-	-	-	-
MaxCutPool	0.56±0.09	0.67±0.09	-	-	-	-
EdgePool	0.45±0.00	0.45±0.00	-	-	-	-
GDPool	**0.43**±0.00	**0.40**±0.00	0.25±0.00	0.23±0.01	6.71±0.08	6.78±0.51

Table 14: Sensitivity to the GFFN expansion ratio for CO_2@0.1 on the hMOF dataset.

GFFN Ratio	MAE	Metric	Layer 1	Layer 2	Layer 3
1	0.125±0.000	Modularity	0.384±0.002	0.382±0.001	0.211±0.002
		K_{eff}	17.759±0.183	9.685±0.079	5.493±0.090
2	0.125±0.002	Modularity	0.388±0.003	0.384±0.005	0.217±0.006
		K_{eff}	17.418±0.305	9.595±0.176	5.534±0.070
4	0.125±0.000	Modularity	0.390±0.003	0.381±0.002	0.218±0.002
		K_{eff}	17.166±0.223	9.565±0.034	5.504±0.050
8	0.126±0.001	Modularity	0.392±0.003	0.382±0.003	0.226±0.006
		K_{eff}	16.921±0.257	9.668±0.194	5.658±0.129