ATLAS: Adaptive Topology-based Learning at Scale for Homophilic and Heterophilic Graphs

Anonymous authors
Paper under double-blind review

Abstract

We present ATLAS (Adaptive Topology-based Learning at Scale for Homophilic and Heterophilic Graphs), a novel graph learning algorithm that addresses two important challenges in graph neural networks (GNNs). First, the accuracy of GNNs degrades when the graph is heterophilic. Second, the iterative feature aggregation limits the scalability of GNNs to large graphs. We address these challenges by extracting topological information about the graph communities at different levels of refinement, concatenating the community assignments to the feature vector, and applying multilayer perceptrons (MLPs) on this new feature vector. By doing so, we inherently obtain the topological data about the nodes and their neighbors without invoking aggregation. Because MLPs are typically more scalable than GNNs, our approach applies to large graphs—without the need for sampling.

Our results, on a wide set of graphs, show that ATLAS has comparable accuracy to baseline methods, with accuracy being as high as 20 percentage points over GCN for heterophilic graphs with negative structural bias and 11 percentage points over MLP for homophilic graphs. Furthermore, we show how multi-resolution community features systematically modulate performance in both homophilic and heterophilic settings, opening a principled path toward explainable graph learning.

1 Introduction

Node classification, a fundamental problem in graph learning, involves identifying labels of nodes in a graph and has wide applications in many domains including social networks, citation networks, recommendation systems, knowledge graphs and bioinformatics (Khemani, 2024; Wu et al., 2019b; Zhou et al., 2021). Accurate classification requires two orthogonal pieces of information–(i) the features at each node, and (ii) the connections between the node and its neighbors. Neural network methods such as Multi-Layer Perceptrons (MLPs) are fast but do not include information about the connections. Graph Neural Networks (GNNs) address this problem by aggregating the features between neighboring nodes, but the process is expensive, and difficult to scale to large graphs. Although the graph structure can be included in the feature vectors using different node embedding techniques (Perozzi et al., 2014; Grover & Leskovec, 2016; Tang et al., 2015), and recently through the use of community detection (Wang et al., 2017; Sun et al., 2019; Kamiński et al., 2024), the issue remains as to how many hops of neighbors should be considered and how fine-grained the communities should be. Taking larger hops or coarse grained community can lead to information smoothing, while taking smaller hops or fine grained communities can lead to information loss. Further, the hypothesis that aggregating features of neighbors can improve accuracy of node classification is only true for homophilic networks (where nodes of similar classes are connected). In heterophilic networks, where the connection between nodes need not imply similarity of class, this strategy actually leads to lower accuracy. Based on these observations, we posit, *matching structural information (i.e. size of hops or communities) with how well it aligns with the classification is necessary for producing accurate results.*

1.1 Related Work

Graph Neural Networks (GNNs) have become a core tool for learning on graphs (Kipf & Welling, 2017; Hamilton et al., 2017). Most algorithms are based on message passing: each layer aggregates transformed neighbor features to form topology-aware embeddings. This implicitly assumes *homophily*(Wu et al., 2019a). The same bias, however, can blur informative distinctions on weakly homophilous or heterophilous graphs (Zhu et al., 2020a).

Sampling-based GCN. Scaling GNNs on large graphs is challenging due to memory requirement, and cost of aggregation. Sampling-based methods construct mini-batches from parts of the graph to approximate full-batch propagation—via node-wise (GraphSAGE (Hamilton et al., 2017)), layer-wise (FastGCN (Chen et al., 2018b)), or subgraph sampling (Cluster-GCN, GraphSAINT (Chiang et al., 2019; Zeng et al., 2020)). However, sampling introduces stochasticity that affects convergence rates and reproducibility (Chen et al., 2018b; Zou et al., 2019).

Community Structure in GNNs. Recently community detection (Newman, 2006; Blondel et al., 2008) has been used to improve performance of GNNS. Information from communities can be integrated into GNN—via line-graph supervision or label-aware aggregation (Chen et al., 2019; 2020). Recent graph rewiring approaches constrain edits using community information (Karhadkar et al., 2023; Jamadandi et al., 2024; Rubio-Madrigal et al., 2025).

Learning on Non-Homophilous Graphs. Recent research also focuses on addressing the problem of learning in graphs that do not exhibit homophily. One approach is to preserve self-features while selectively injecting neighborhood signals (H2GCN (Zhu et al., 2020b), GloGNN (Li et al., 2022)). Another approach is to negate misleading neighbors (GPR-GNN (Chien et al., 2021), FAGCN (Bo et al., 2021)). Some methods leverage higher-order structures (MixHop (Abu-El-Haija et al., 2019)); other methods try to identify homophilic and heterophilic signals based on kernel features and spectral filters (JacobiConv (Wang & Zhang, 2022), BernNet (He et al., 2021) GBK-GNN (Du et al., 2022)).

Graph rewiring is another approach to addressing heterophily. Graph rewiring focuses on adding edges between similar nodes and removing edges between edges that are dissimilar. Several methods can be used for rewiring, including adding edges within intra-community nodes or nodes with similar features, deleting edges between inter-community nodes, and rewring to maximize the spectral distance (Topping et al., 2022; Karhadkar et al., 2023; Gasteiger et al., 2019; Bi et al., 2024; Rubio-Madrigal et al., 2025; Banerjee et al., 2022).

1.2 Our Contribution.

Most of the curent research either focuses primarily on homophilic graphs, or the processes to address the heterophilic graphs are require expensive operations, such as signal identification/modification, rewiring or spectral gap maximization. These methods cannot efficiently scale to large graphs. Our **primary contribution** is to develop **A**daptive **T**opology -based **L**earning at **S**cale **(ATLAS)** *, a novel graph learning algorithms that can produce high-accuracy results for both homophilic and heterophilic graphs. ATLAS is based on a *simple but powerful technique of refining communities in networks to match the degree of homophily.*

Rationale. Our algorithm is based on quantifying homophily through the lens of normalized mutual information (NMI). Given two partitions of the same set of elements NMI measures how well the partitions correspond to each other. If we consider one partition as the communities in the graph, and the other partition as labels, then NMI provides a measure for the degree of homophily in the graph. ATLAS focuses on refining/coarsening communities to identify the region of highest NMI—which will correspond to the highest accuracy. Figure 1 provides an overview of ATLAS.

Our specific contributions are;

1. **Theory.** We provide a theoretical analysis of how refining communities changes in NMI (Section 2).

*Apart from the acronym, the name ATLAS is to convey our method can handle different degrees of homophily, similar to how an atlas encompasses all different countries.

2. **Algorithm.** Based on this mathematical understanding we develop our algorithm ATLAS (Section 3).
3. **Experiments.** Provide extensive empirical evaluations by comparing ATLAS across a mix of 13 (8 medium size and 5 large) homophilic and heterophilic graphs, and 12 (7) GNN/MLP-based algorithms for medium sized (large) graphs (Section 4).

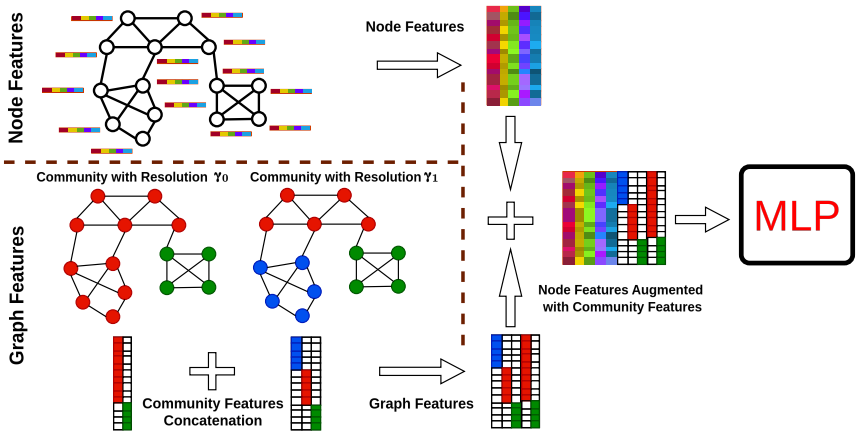


Figure 1: Overview of the community-augmented feature learning pipeline. Community assignments at multiple resolutions are one-hot encoded, projected, concatenated with node features, and input to an MLP for classification.

2 Theoretical Analysis

We mathematically show how refining communities leads to changes in NMI. We define some terms that will help us in the analysis. The proofs of the theorems are given in the appendix.

Let N be the set of nodes. Let $P = \{P_1, \ldots, P_k\}$ be a partition of N; i.e.

$$P_i \neq \varnothing, \quad P_i \cap P_j = \varnothing \ (i \neq j), \quad \text{and} \quad \bigcup_{i=1}^{k} P_i = N.$$

Let $S = \{S_1, \ldots, S_m\}$ be another partition of N. We say S *is a refinement of* P (denoted as $S \preceq P$) iff every block of S is contained in some block of P. Formally:

$$S \preceq P \iff \forall\, S_j \in S\ \exists\, P_i \in P \text{ such that } S_j \subseteq P_i.$$

Normalized mutual information (NMI) is a popular measure to quantify alignment between two partitions. Given two partition P and Q, over a set of N elements and $n_{ij} = |P_i \cap Q_j|, n_i = |P_i|, n_j = |Q_j|$ their normalized mutual information is given as;

$$\text{NMI}(P, Q) \ = \ \frac{2I(P;Q)}{H(P) + H(Q)}$$

$I(P;Q) \ = \ \sum_{i=1}^{k} \sum_{j=1}^{m} \frac{n_{ij}}{N} \log\left(\frac{N\, n_{ij}}{n_i\, n_j}\right)$ is the mutual information between partitions P and Q. This quantity measures how much information is shared between the partitions P and Q. The higher the value, the better the alignment between the partitions. $H(P) = -\sum_{i=1}^{k} \frac{n_i}{N} \log\left(\frac{n_i}{N}\right)$, is the entropy of partition P. $H(Q)$ is defined similarly. The entropy measures the distribution of points in each partition. Low entropy means data is concentrated in few clusters, and is indicative of good clustering.

The value of NMI ranges from 1 (indicating complete alignment between partitions) to close to 0 (indicating complete mismatch between partitions). NMI is high if the partitions are well matched ($I(P, Q)$ is high), and entropy is low ($H(P)$, $H(Q)$ is low).

Lemma 1 (Refinement does not decrease mutual information)**.** *Let L be labels and C a community partition. Let C' be a refinement of C, i.e., $C' \preceq C$. Then $I(L; C') \geq I(L; C)$*

Lemma 2 (Refinement does not decrease entropy)**.** *Let C a community partition. Let C' be a refinement of C, i.e., $C' \preceq C$. Then $H(C') \geq H(C)$*

Based on Lemma 1 and Lemma 2 we see that while refinement improves the mutual information leading to better alignment, it also increases the entropy leading to more noise or uncertainty. The condition at whch NMI will increase is given by Theorem 1.

Theorem 1 (NMI Refinement Condition)**.** *Let L be labels; C a community partition. Let C' be a refinement of C, i.e., $C' \preceq C$. Then $NMI(C'; L) > NMI(C; L)$ if and only if $\frac{\Delta I}{\Delta H} > \frac{NMI(C;L)}{2}$; where $\Delta I = I(C'; L) - I(C; L)$ and $\Delta H = H(C'; L) - H(C; L)$*

Theorem 1 states that a partition refinement improves the normalized mutual information with respect to labels if and only if the mutual information gain per unit of entropy increase exceeds half the original normalized mutual information value.

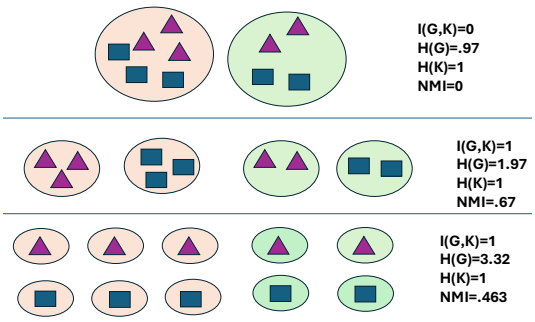


Figure 2: Effect of refinement on NMI. Initially when clusters have mixed items, NMI is low. The first refinement matches the items and clusters, increasing the NMI. Further refinement does not improve the alignment (mutual information), but increases the spread (entropy), thus decreasing NMI.

3 Methodology

The theorems in Section 2 are based on idealized condition, where refined communities are perfect subset of the original communities. In practice, refinement in communities is achieved by modifying a resolution parameter. Although higher resolution leads to smaller communities, due to inherent non-determinism of community detection methods, the smaller communities may not be exact subsets.

Preprocessing. Communities in networks are often hierarchical, so we treat the resolution parameter γ as the hierarchy level. We start from two initial resolutions (0.5 and 1.0) and set three hyperparameters: a gap threshold $\Delta_{\max}$, a minimum modularity τ, and a small target-drop range $[a, b]$. Let γ_1 and γ_2 be two consecutive resolution parameters, with community sets $\mathbf{c}^{(\gamma_1)}$ and $\mathbf{c}^{(\gamma_2)}$ and modularities $Q^{(\gamma_1)}$ and $Q^{(\gamma_2)}$; we define the modularity gap as $\Delta Q = |Q^{(\gamma_2)} - Q^{(\gamma_1)}|$. At each iteration, we sort the tested resolutions and examine consecutive pairs. If the gap between a pair exceeds $\Delta_{\max}$, we find their midpoint (interpolation). Otherwise, we extrapolate beyond the current maximum by estimating the local slope of modularity with respect to resolution and taking a small forward step expected to reduce modularity by a random amount drawn from the drop range. Once the new γ is obtained we compute the communities at that value. The loop stops when the latest modularity falls below τ or no new resolution is produced. The procedure returns the retained resolutions and their corresponding community assignments; see Algorithm 1 in appendix.

Figure 3 illustrates this process of generating sets of resolution paramters. In Step 1 and Step 2, the communities are obtained with initial resolutions γ_0 and γ_1. Step 3 shows an example of interpolation to γ_2, and steps 4 and 5 show extrapolation to finer resolutions at γ_3 and γ_4. Note that $\gamma_0 > \gamma_2 > \gamma_1 > \gamma_3 > \gamma_4$, and the communities for one resolution parameter is the refinement of the previous larger resolution parameter.

Feature Augmentation. For a given resolution parameter γ, the communities be $\mathbf{c}^{(\gamma)} \in \{1, \ldots, k_\gamma\}$, and and each node is assigned to one of the communities in k_γ. This assignment is represented as a one-hot encoded matrix $\mathbf{H}^{(\gamma)}$(equation 3). To reduce dimensionality, each one-hot matrix is projected into a dense embedding space using a trainable weight matrix $\mathbf{W}^{(\gamma)}$ (equation 4). The embeddings from all resolutions are concatenated to form $\mathbf{E}$ (equation 5), which is then further concatenated with the original features $\mathbf{X}$ to yield the augmented feature matrix $\mathbf{Z}$. The augmented feature matrix $\mathbf{Z}$ is fed to an MLP f_θ to produce logits; a task-dependent function ϕ (e.g., softmax or elementwise sigmoid), applied row-wise, converts them to probabilities $\hat{\mathbf{Y}}$ (equation 7).

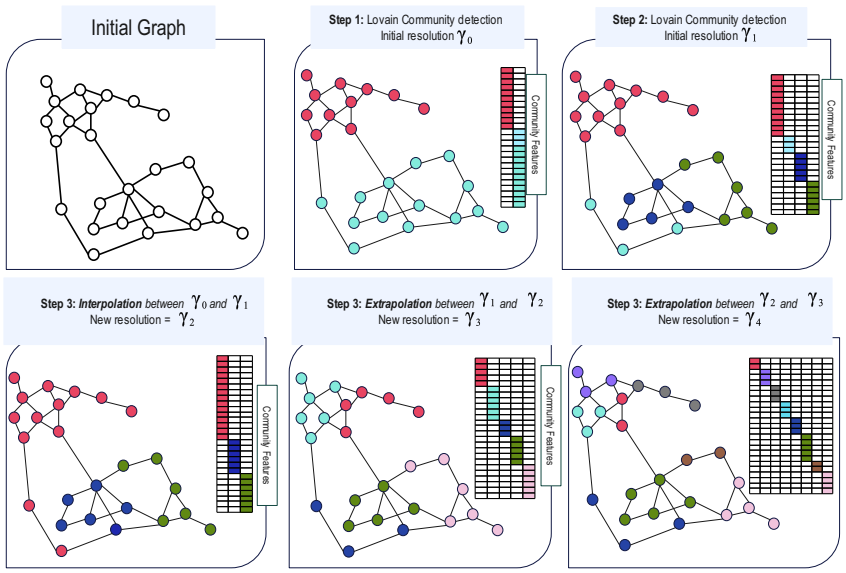


Figure 3: Illustration of the Adaptive Resolution Search Process. The resolution limits, $\gamma_4 < \gamma_3 < \gamma_1 < \gamma_2 < \gamma_0$, and the communities $C^{\gamma_0} \preceq C^{\gamma_2} \preceq C^{\gamma_1} \preceq C^{\gamma_3} \preceq C^{\gamma_4}$.

.

$$\mathbf{X} \in \mathbb{R}^{n \times d}, \quad \Gamma = \{\gamma_1, \ldots, \gamma_T\} \tag{1}$$

$$\mathbf{c}^{(\gamma)} = \text{DetectCommunity}(G, \gamma), \quad \mathbf{c}^{(\gamma)} \in \{1, \ldots, k_\gamma\}^n \tag{2}$$

$$\mathbf{H}^{(\gamma)} = \text{OneHot}(\mathbf{c}^{(\gamma)}) \in \{0,1\}^{n \times k_\gamma} \tag{3}$$

$$\mathbf{E}^{(\gamma)} = \mathbf{H}^{(\gamma)} \mathbf{W}^{(\gamma)}, \quad \mathbf{W}^{(\gamma)} \in \mathbb{R}^{k_\gamma \times d_c} \tag{4}$$

$$\mathbf{E} = \Big\|_{t=1}^{T} \mathbf{E}^{(\gamma_t)} \in \mathbb{R}^{n \times (T d_c)} \tag{5}$$

$$\mathbf{Z} = [\mathbf{X} \parallel \mathbf{E}] \in \mathbb{R}^{n \times (d + T d_c)} \tag{6}$$

$$\hat{\mathbf{Y}} = \phi\big(f_\theta(\mathbf{Z})\big) \in [0,1]^{n \times C} \tag{7}$$

Complexity Analysis. We compare computational and memory complexities of representative scalable GNN frameworks with our approach in Table 1. ATLAS performs Louvain clustering in $O(T\|A\|_0)$ in the preprocessing step, keeps a single augmented feature buffer, and trains with per-epoch time $O\big(L_{ff} N (D+Td_c)^2\big)$ and memory $O\big(bL_{ff}(D+Td_c)\big)$, enabling simple i.i.d. node mini-batching without neighborhood expansion or graph-dependent batching heuristics. ATLAS performs adjacency-free inference: with fixed augmented features of dimension $D+Td_c$, prediction is a forward pass with complexity $O\big(N(D+Td_c)^2\big)$.

4 Empirical Evaluation

In this section, we provide the empirical results comparing ATLAS with other graph learning methods. Our experiments focus on answering the following *research questions*;

Q1. How accurate is ATLAS compared to baseline methods over graphs with different degrees of homophily?
Q2. How well can ATLAS scale to large graphs, while maintaining high accuracy?
Q3. How does degree of homophily affect the optimal refinement level?

Datasets. We use 8 medium-size graphs (Cora, PubMed, Tolokers, Squirrel-filtered, Chameleon-filtered, Amazon-ratings, Actor, Roman-empire) and 5 large graphs (Flickr,

Table 1: Complexity comparison. N = #nodes, $\|A\|_0$ = #edges, D = feature dim, L = #message-passing layers, L_{ff} = #feed-forward layers, b = batch size, r = sampled neighbors (or filter size), T = #resolutions, d_c = community-embedding dim.

Method	**Preprocessing**	**Training per epoch (time)**	**Memory**
GCN (full-batch)	–	$O(L\,\|A\|_0 D + LND^2)$	$O(LND + LD^2)$
FastGCN	–	$O(r^L ND^2)$	$O(b\,r^L D + LD^2)$
S-GCN (VR-GCN)	–	$O(rLND^2)$	$O(LND)$
ClusterGCN	$O(\|A\|_0)$	$O(L\,\|A\|_0 D + LND^2)$	$O(bLD + LD^2)$
GraphSAINT	$O(kN)$	$O(L\,\|A\|_0 D + LND^2)$	$O(bLD)$
ATLAS	$O(T\|A\|_0)$	$O(L_{ff} N(D + Td_c)^2)$	$O(bL_{ff}(D + Td_c))$

Table 2: Eight-benchmark comparison across homophily regimes. Metrics are mean accuracy (%) ± standard deviation, *except* Tolokers, which reports ROC–AUC (%). Baseline heterophily-oriented model results are from Platonov et al. (2023); Luan et al. (2024). Bottom rows report ATLAS improvements over baselines (absolute percentage points).

	High structural bias		**Low structural bias**				**Negative structural bias**	
Model	Cora h_e=0.810	Tolokers h_e=0.595	PubMed h_e=0.802	Squirrel-filtered h_e=0.207	Chameleon-filtered h_e=0.236	Amazon-ratings h_e=0.380	Actor h_e=0.216	Roman-empire h_e=0.047
MLP(2L)	75.44 ± 1.97	72.97 ± 0.90	87.25 ± 0.41	34.29 ± 3.34	36.00 ± 4.69	39.83 ± 0.48	34.96 ± 0.71	65.58 ± 0.34
GCN	87.01 ± 1.04	74.93 ± 1.32	86.71 ± 0.42	32.70 ± 1.73	37.11 ± 3.04	42.78 ± 0.14	28.49 ± 0.91	45.68 ± 0.38
SAGE	87.50 ± 0.87	80.95 ± 0.92	88.42 ± 0.55	33.32 ± 1.75	38.83 ± 4.26	44.67 ± 0.51	34.08 ± 1.07	76.21 ± 0.65
GAT	87.74 ± 0.88	75.31 ± 1.35	86.18 ± 0.64	32.61 ± 2.06	37.18 ± 3.44	43.25 ± 0.85	29.11 ± 1.23	47.16 ± 0.66
H_2GCN	87.52 ± 0.61	73.35 ± 1.01	87.78 ± 0.28	35.10 ± 1.15	26.75 ± 3.64	36.47 ± 0.23	38.85 ± 1.17	60.11 ± 0.52
LinkX	82.62 ± 1.44	81.15 ± 1.23	88.12 ± 0.47	42.34 ± 4.13	40.10 ± 2.21	52.66 ± 0.64	35.64 ± 1.36	56.15 ± 0.93
GPR-GNN	79.51 ± 0.36	72.94 ± 0.97	85.07 ± 0.09	38.95 ± 1.99	39.93 ± 3.30	44.88 ± 0.34	39.30 ± 0.27	64.85 ± 0.27
FSGNN	87.51 ± 1.21	82.76 ± 0.61	90.11 ± 0.43	35.92 ± 1.32	40.61 ± 2.97	52.74 ± 0.83	37.65 ± 0.79	79.92 ± 0.56
GloGNN	87.67 ± 1.16	73.39 ± 1.17	90.32 ± 0.54	35.11 ± 1.24	25.90 ± 3.58	36.89 ± 0.14	39.65 ± 1.03	59.63 ± 0.69
FAGCN	88.85 ± 1.36	77.75 ± 1.05	89.98 ± 0.54	41.08 ± 2.27	41.90 ± 2.72	44.12 ± 0.30	31.59 ± 1.37	65.22 ± 0.56
GBK-GNN	87.09 ± 1.52	81.01 ± 0.67	88.88 ± 0.44	35.51 ± 1.65	39.61 ± 2.60	45.98 ± 0.71	38.47 ± 1.53	74.57 ± 0.47
JacobiConv	89.61 ± 0.96	68.66 ± 0.65	89.99 ± 0.39	29.71 ± 1.66	39.00 ± 4.20	43.55 ± 0.48	37.48 ± 0.76	71.14 ± 0.42
ATLAS (Ours)	**87.09 ± 1.62**	**81.39 ± 0.76**	**88.29 ± 0.62**	**38.30 ± 2.31**	**40.17 ± 4.06**	**53.15 ± 0.61**	**38.07 ± 0.93**	**66.22 ± 0.53**
ATLAS−MLP (pp)	+11.65	+8.42	+1.04	+4.01	+4.17	+13.32	+3.11	+0.64
ATLAS−GCN (pp)	+0.08	+6.46	+1.58	+5.60	+3.06	+10.37	+9.58	+20.54
ATLAS−Average (pp)	+1.42	+5.13	+0.06	+2.75	+3.26	+9.17	+2.63	+2.37

Reddit, Yelp, Amazon-Products, OGBN-Products). We report accuracy for all medium-size datasets *except* Tolokers, which is evaluated by ROC-AUC; for large graphs, we report F1-micro on Flickr/Reddit/Yelp/Amazon-Products and accuracy on OGBN-Products. Complete statistics of the datasets are given in Appendix Tables 9 and 8.

Baselines. We group baselines by modeling regime and map them to the research questions. *Q1 (homophily-regime). Homophilic*: GCN (Kipf & Welling, 2017), GraphSAGE (Hamilton et al., 2017), GAT (Veličković et al., 2018). *Heterophily-oriented*: H_2GCN (Zhu et al., 2020a), LinkX (Lim et al., 2021), GPR-GNN (Chien et al., 2021), FSGNN (Maurya et al., 2022), GloGNN (Li et al., 2022), FAGCN (Bo et al., 2021), GBK-GNN (Du et al., 2022), JacobiConv (Wang & Zhang, 2022). *Q2 (scalability). Sampling-based*: GraphSAGE (Hamilton et al., 2017), FastGCN (Chen et al., 2018b), S-GCN (variance-reduced) (Chen et al., 2018a), ClusterGCN (Chiang et al., 2019), GraphSAINT (Zeng et al., 2020). Description of these methods are provided in the Appendix.

We use an L-layer MLP with hidden width d_{hid} and dropout rate p. Each of the first $L-1$ layers applies *Linear (with bias)* → *LayerNorm* → *GELU* → *Dropout*. The final layer is a *Linear* classifier to C classes.

4.1 Q1: Accuracy Across Homophily Regimes

Table 2 shows results for the eight medium-sized benchmarks. On average, ATLAS shows as much as 20 percentage points over GCN for graphs with negative structural bias and as much as 11 percentage points over MLP for graphs with high structural bias. Overall, ATLAS closes the MLP→GNN gap and delivers results that are consistently comparable to the best model on each dataset.

Table 3: Large-graph performance. F1-micro for Flickr/Reddit/Yelp/Amazon-Products; accuracy for OGBN-Products. Baselines from Zeng et al. (2020); Hu et al. (2020). Bottom rows report ATLAS improvements over MLP and over GCN (absolute units).

	High structural bias		**Low structural bias**	**Negative structural bias**	
Method	Reddit h_e=0.756	ogbn-products h_e=0.808	Flickr h_e=0.319	Yelp h_e=0.809	AmazonProducts h_e=0.116
MLP	0.7435 ± 0.0016	0.6106 ± 0.0008	0.4717 ± 0.0011	0.6546 ± 0.0011	0.8204 ± 0.0002
GCN	0.9330±0.0000	0.7564±0.0021	0.4920±0.0030	0.3780±0.0010	0.2810±0.0050
GraphSAGE	0.9530±0.0010	0.8061±0.0016	0.5010±0.0130	0.6340±0.0060	0.7580±0.0020
FastGCN	0.9240±0.0010	0.7346±0.0020	0.5040±0.0010	0.2650±0.0530	0.1740±0.0210
S-GCN	0.9640±0.0010	0.7590±0.0000	0.4820±0.0030	0.6400±0.0020	—
ClusterGCN	0.9540±0.0010	0.7862±0.0061	0.4810±0.0050	0.6090±0.0050	0.7590±0.0080
GraphSAINT	0.9660±0.0010	0.7536±0.0034	0.5110±0.0010	0.6530±0.0030	0.8150±0.0010
ATLAS (Ours)	**0.9574 ± 0.0004**	**0.7865 ± 0.0053**	**0.5064 ± 0.0017**	**0.6546 ± 0.0011**	**0.8204 ± 0.0002**
ATLAS−MLP	+0.2139	+0.1759	+0.0347	+0.0000	+0.0000
ATLAS−GCN	+0.0244	+0.0301	+0.0144	+0.2766	+0.5394
ATLAS−Average	+0.0378	+0.0427	+0.0146	+0.1069	+0.2192

Table 4: Computation time breakdown (in seconds) on OGBN-Products. We report preprocessing, average per-epoch training, and inference time. Values are mean ± std over repeated runs. For methods with heavy one-time preprocessing, we separate that cost.

Model	Preprocessing Time	Per-epoch Train Time	Inference Time
GCN	—	2.0395 ± 0.0006	0.9220 ± 0.0010
Cluster-GCN	168.754 ± 1.777 (one-time)	4.017 ± 0.164	82.837 ± 0.622
GraphSAINT	3.770 ± 0.159 (per epoch)	0.751 ± 0.046	66.445 ± 0.517
ATLAS (Ours)	391.894 ± 14.387 (one-time)	0.181 ± 0.0058	0.526 ± 0.0038

Table 5: Preprocessing (community detection), training, and inference times. Values are mean ± sample std in seconds across runs.

Dataset	Preprocessing Time	Per-epoch Train Time	Inference Time
Reddit	84.904 ± 2.764	0.143 ± 0.002	0.150 ± 0.005
Flickr	6.800 ± 1.741	0.241 ± 0.005	0.056 ± 0.012
Yelp	15.842 ± 0.007	2.670 ± 0.007	1.613 ± 0.016
AmazonProducts	72.270 ± 1.409	6.073 ± 0.039	3.056 ± 0.019

4.2 Q2: Efficiency and Scalability on Large Graphs

Accuracy. Table 3 results demonstrate that ATLAS is scalable and delivers competitive accuracy on million-scale graphs while maintaining clear gains over MLP and GCN. On average, ATLAS shows as much as +.53 points over GCN for graphs with negative structural bias and as much as +.21 percentage points over MLP for graphs with high structural bias.

Convergence. ATLAS converges rapidly and stably across large graphs: training loss decreases smoothly, and validation performance (F1-micro for Flickr/Reddit/Yelp/Amazon-Products; accuracy for OGBN-Products) plateaus early with a small train–validation gap. The curves exhibit no late-epoch degradation and remain stable after convergence (see Fig. 4).

Efficiency. Table 4 compares preprocessing, per-epoch training, and inference times on OGBN-Products. ATLAS requires a higher one-time preprocessing cost yet attains competitive training speed and the **best inference time**, offering a balanced trade-off between training scalability and evaluation efficiency. Table 5 shows the time for the other large graphs [†].

[†]We could not compare with the baselines for these, as the optimal hyper-parameters are not known.

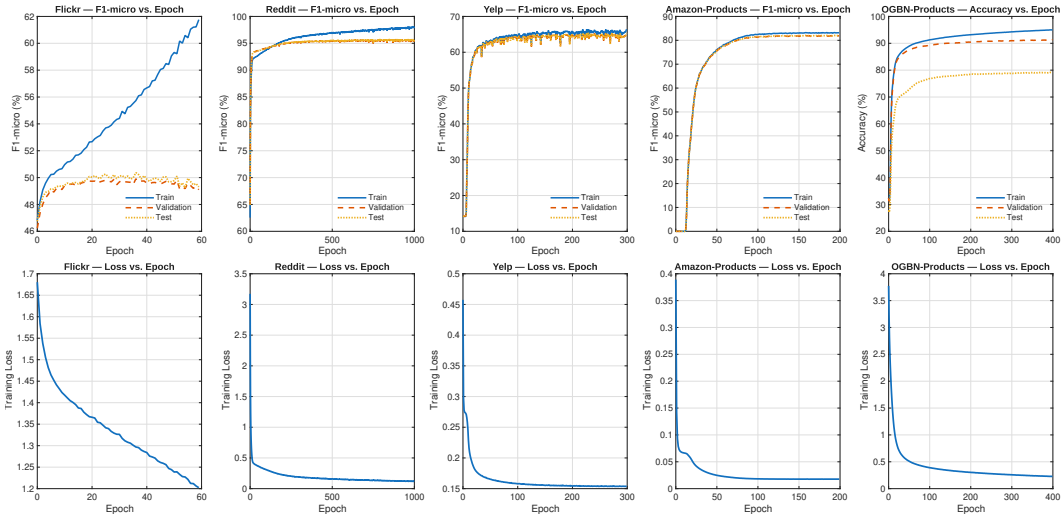


Figure 4: The convergence landscape of ATLAS.

4.3 Q3: Relation of Degree of Homophily to Refinement Level

We quantify the level of community refinement by the minimum modularity threshold $Q_{\min}$. Large $Q_{\min}$ preserves only coarse partitions; lowering $Q_{\min}$ progressively adds medium and fine partitions, yielding a multi-scale representation. We group graphs based on how well the communities align with node labels as follows;

- *High structural bias* (e.g., Cora, Tolokers): community aligns with labels; decreasing $Q_{\min}$ (adding more scales) improves the metric (accuracy for Cora; ROC–AUC for Tolokers) until fine partitions yield marginal gains or mild saturation. See Figure 5; top left and bottom left.
- *Low structural bias* (e.g., Amazon-Ratings, Chameleon-filtered): communities provide limited label-aligned signal; coarse partitions give small gains, while finer ones add little beyond saturation. See Figure 5; top center and bottom center.
- *Negative structural bias* (e.g., Actor, Roman-Empire): community misaligns with labels; adding finer partitions introduces misleading locality and degrades performance relative to the high-$Q_{\min}$ (feature-dominant) regime. See Figure 5; top right and bottom right.

Example (Cora). For a specific example, consider the Cora graph (Fig. 5 top left; Table 7), $Q_{\min} \in \{1.0, 0.9\}$ retains no partitions and accuracy is 76.61%. At $Q_{\min} = 0.8$, two coarse partitions are added and accuracy rises to 79.93%; at 0.7, adding one medium partition yields 83.66%; at 0.6, two finer partitions reach 86.50%. Accuracy peaks at 88.10% for $Q_{\min} = 0.1$, then slightly drops at 0.0 after adding the most fragmented partition, indicating diminishing returns from very small communities. The effect of cumulative community features depends on a dataset's structural bias (more details in Table 7 in appendix).

Effect of Modularity Gap. The set of refinement levels depends heavily on the modularity gap. For graphs with high structural bias (Figure 6(a)), accuracy is flat-to-low when the avg. modularity gap is tiny ($g \in [0, 0.05]$), climbs to a brief sweet spot around $g \approx 0.06{-}0.09$, and then tails off past ~ 0.10 indicating increase in entropy outweights gain in mutual information. For graphs with low structural bias (Figure 6(b)) increasing modularity gap does not change the accuracy, as the communities are already misaligned, and gain in mutual information is not significant.

5 Conclusion and Future Work

We presented ATLAS, a community-augmented learning framework that enriches node features with multi-resolution Louvain embeddings and trains a compact MLP classifier. An

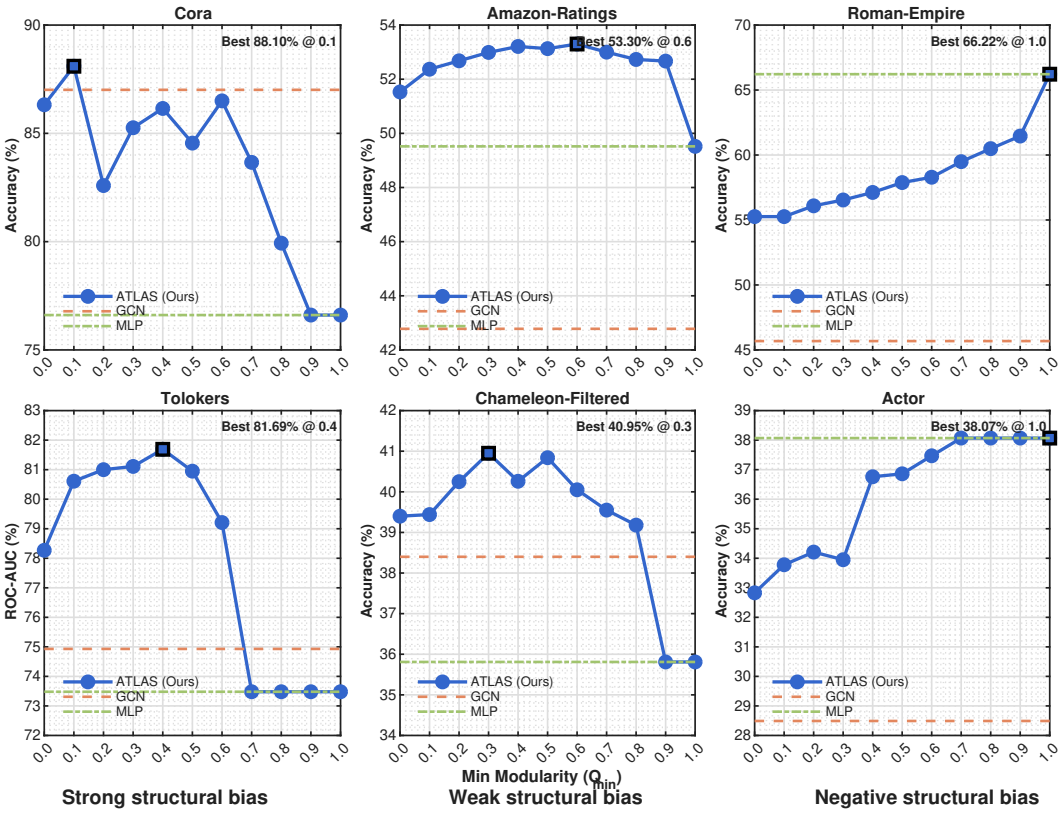
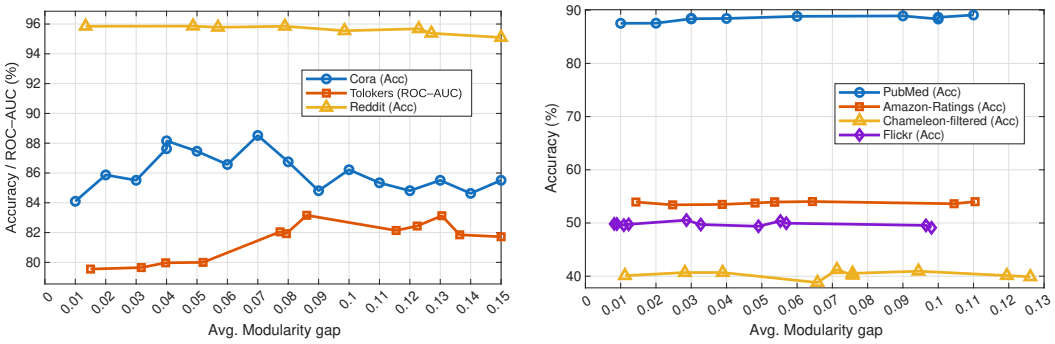


Figure 5: Effect of cumulative community-derived features gated by minimum modularity in homophilic (Strong structural bias, left), Benign heterophilic (Weak structural bias, middle) and Malignant heterophilic (Negative structural bias, right) settings.



(a) **High structural bias datasets**. (b) **Low structural bias datasets**.

Figure 6: Accuracy/ROC–AUC vs. average modularity gap.

adaptive resolution search, governed by $Q_{\min}$ and ΔQ, selects a small set of informative resolutions, balancing coverage with cost. Across Q1 (homophily-regime benchmarks) and Q2 (large graphs), ATLAS attains competitive or superior accuracy relative to homophilic GNNs and heterophily-oriented models, while exhibiting fast, stable convergence and a favorable training footprint once preprocessing is complete.

In future we aim to reduce preprocessing by improving resolution selection. A complementary direction is community-guided graph rewiring: using the discovered communities to propose sparse, label-aware edge edits that amplify useful intra-/inter-community signals. and further improve accuracy.

Ethics Statement. We confirm that we have adhered to the to the ICLR Code of Ethics.

Use of Generative AI. We have used generative Ai to polish the writing, and to check that the proofs of the theorem and lemma are correct and concise.

Reproducibility Statement. Our source code is available at `https://github.com/atlaspaper16/ATLAS`. This github is created through an anonymous account, and thus does not violate the double blind policy.

References

Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. MixHop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, volume 97 of *Proceedings of Machine Learning Research*, pp. 21–29. PMLR, 2019. URL `https://proceedings.mlr.press/v97/abu-el-haija19a.html`.

Pradeep Kr. Banerjee, Mitchell Black, Francesco Di Giovanni, Gustavo Montes, Yuval Peres, and Guido Montúfar. Oversquashing in graph neural networks through the lens of information contraction. In *2022 60th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2022. URL `https://allerton.csl.illinois.edu/files/2022/12/2022-118-paper_9389.pdf`.

Wendong Bi, Lun Du, Qiang Fu, Yanlin Wang, Shi Han, and Dongmei Zhang. Make heterophily graphs better fit gnn: A graph rewiring approach. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2024. URL `https://www.computer.org/csdl/journal/tk/2024/12/10634240/1ZlBd5BHIWI`. Deep Heterophily Graph Rewiring (DHGR).

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. doi: 10.1088/1742-5468/2008/10/P10008.

Deyu Bo, Xiao Wang, Chuan Shi, and Huawei Shen. Beyond low-frequency information in graph convolutional networks. In *AAAI Conference on Artificial Intelligence*, 2021.

Hao Chen, Yue Xu, Feiran Huang, Zengde Deng, Wenbing Huang, Senzhang Wang, Peng He, and Zhoujun Li. Label-aware graph convolutional networks. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management*, CIKM '20, pp. 1977–1980, 2020. doi: 10.1145/3340531.3412139.

Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pp. 942–950, 2018a. URL `https://proceedings.mlr.press/v80/chen18p.html`.

Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: Fast learning with graph convolutional networks via importance sampling. In *International Conference on Learning Representations (ICLR)*, 2018b. URL `https://openreview.net/forum?id=rytstxWAW`.

Zhengdao Chen, Lisha Li, and Joan Bruna. Supervised community detection with line graph neural networks. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=H1g0Z3A9Fm`.

Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, pp. 257–266, 2019. doi: 10.1145/3292500.3330925. URL `https://arxiv.org/abs/1905.07953`.

Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. In *International Conference on Learning Representations*, 2021.

Lun Du, Xiaozhou Shi, Qiang Fu, Xiaojun Ma, Hengyu Liu, Shi Han, and Dongmei Zhang. Gbk-gnn: Gated bi-kernel graph neural networks for modeling both homophily and heterophily. In *Proceedings of the ACM Web Conference (WWW)*, pp. 1551–1560, 2022. doi: 10.1145/3485447.3512201. URL https://dl.acm.org/doi/10.1145/3485447.3512201.

Johannes Gasteiger, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. URL https://papers.neurips.cc/paper/9490-diffusion-improves-graph-learning.pdf.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, 2016. doi: 10.1145/2939672.2939754. URL https://cs.stanford.edu/~jure/pubs/node2vec-kdd16.pdf.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017. URL https://arxiv.org/abs/1706.02216.

Mingguo He, Zhe Wei, Bolin Ding, Yaliang Li, and Ji Liu. Bernnet: Learning arbitrary graph spectral filters via bernstein approximation. In *Advances in Neural Information Processing Systems*, 2021.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowei Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/fb60d411a5c5b72b2e7d3527cfc84fd0-Abstract.html.

Adarsh Jamadandi, Celia Rubio-Madrigal, and Rebekka Burkholz. Spectral graph pruning against over-squashing and over-smoothing. In *Advances in Neural Information Processing Systems*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/140aac600566125915df7e74ff538f66-Paper-Conference.pdf.

Bogumił Kamiński, Paweł Prałat, François Théberge, and Sebastian Zając. Predicting properties of nodes via community-aware features. *Social Network Analysis and Mining*, 14(117), 2024. doi: 10.1007/s13278-024-01281-2. URL https://link.springer.com/article/10.1007/s13278-024-01281-2.

Kedar Karhadkar, Pradeep Kr. Banerjee, and Guido Montúfar. FoSR: First-order spectral rewiring for addressing oversquashing in GNNs. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=3YjQfCLdrzz.

B. Khemani. A review of graph neural networks: concepts, architectures, and applications. *Journal of Big Data*, 2024. doi: 10.1186/s40537-023-00876-4. URL https://journalofbigdata.springeropen.com/articles/10.1186/s40537-023-00876-4.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=SJU4ayYgl.

Xiang Li, Renyu Zhu, Yao Cheng, Caihua Shan, Siqiang Luo, Dongsheng Li, and Weining Qian. Finding global homophily in graph neural networks when meeting heterophily. In *International Conference on Machine Learning*, 2022.

Derek Lim, Felix Hohne, Xiuyu Li, Sijia Linda Huang, Vaishnavi Gupta, Omkar Bhalerao, and Ser-Nam Lim. Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pp. 30471–30483, 2021. URL https://papers.neurips.cc/paper_files/paper/2021/file/ae816a80e4c1c56caa2eb4e1819cbb2f-Paper.pdf.

Sitao Luan, Qincheng Lu, Chenqing Hua, Xinyu Wang, Jiaqi Zhu, Xiao-Wen Chang, Guy Wolf, and Jian Tang. Are heterophily-specific gnns and homophily metrics really effective? evaluation pitfalls and new benchmarks. *arXiv preprint arXiv:2409.05755*, 2024.

Sunil Kumar Maurya, Xin Liu, and Tsuyoshi Murata. Simplifying approach to node classification in graph neural networks. *Knowledge-Based Systems*, 246:108586, 2022.

M. E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006. doi: 10.1073/pnas.0601602103.

Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Li, Yiming Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1e2agrFvS.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *KDD*, 2014. doi: 10.1145/2623330.2623732.

Oleg Platonov, Denis Kuznedelev, Michael Diskin, Artem Babenko, and Liudmila Prokhorenkova. A critical look at the evaluation of GNNs under heterophily: Are we really making progress? *arXiv preprint arXiv:2302.11640*, 2023. URL https://arxiv.org/abs/2302.11640.

Celia Rubio-Madrigal, Adarsh Jamadandi, and Rebekka Burkholz. GNNs getting ComFy: Community and feature similarity guided rewiring. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=g6v09VxgFw.

Prithviraj Sen, Gal Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93–106, 2008. URL https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/2157.

Fan-Yun Sun, Meng Qu, Jordan Hoffmann, Chin-Wei Huang, and Jian Tang. vgraph: A generative model for joint community detection and node representation learning. In *NeurIPS*, 2019. URL https://papers.neurips.cc/paper/8342-vgraph-a-generative-model-for-joint-community-detection-and-node-representation-learning.pdf.

Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. LINE: Large-scale information network embedding. In *WWW*, 2015. doi: 10.1145/2736277.2741093.

James Topping, Francesco Di Giovanni, Benjamin P. Chamberlain, Xiaowen Dong, and Michael M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. In *International Conference on Learning Representations (ICLR)*, 2022. URL https://openreview.net/forum?id=7UmjRGzp-A.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations (ICLR)*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. Community preserving network embedding. In *AAAI*, 2017. URL https://ojs.aaai.org/index.php/AAAI/article/view/10488.

Xiyuan Wang and Muhan Zhang. How powerful are spectral graph neural networks. In *International Conference on Machine Learning*, 2022.

Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr., Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying graph convolutional networks. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, volume 97 of *Proceedings of Machine Learning Research*, pp. 6861–6871. PMLR, 2019a. URL https://proceedings.mlr.press/v97/wu19e.html.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *arXiv:1901.00596*, 2019b. URL https://arxiv.org/abs/1901.00596.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. In *International Conference on Learning Representations (ICLR)*, 2020. URL https://openreview.net/forum?id=BJe8pkHFwS.

Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2021. doi: 10.1016/j.aiopen.2021.01.001.

Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020a. URL https://proceedings.neurips.cc/paper/2020/hash/58ae23d878a47004366189884c2f8440-Abstract.html.

Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. In *NeurIPS*, 2020b. URL https://proceedings.neurips.cc/paper/2020/file/58ae23d878a47004366189884c2f8440-Paper.pdf.

Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. Layer-dependent importance sampling for training deep and large graph convolutional networks. In *Advances in Neural Information Processing Systems*, volume 32, pp. 1125–1136, 2019. URL https://dl.acm.org/doi/10.5555/3454287.3455296.

6 Appendix

Compute environment. All experiments were run on a server with 1× NVIDIA A40 (45 GiB) GPU, 32 vCPUs, 2× Intel Xeon Silver 4309Y @ 2.80 GHz, and 503 GiB RAM. Software stack: Python 3.10.18; PyTorch 2.4.0+cu124 (CUDA 12.4); PyTorch Geometric 2.6.1.

6.1 Theoretical Proofs

Lemma. 1. *Let L be labels and C a community partition. Let C' be a refinement of C, i.e., $C' \preceq C$. Then $I(L;C') \geq I(L;C)$*

Proof. Let total number of elements be n. Then based on the definitions of $I(P,Q)$ in Section 2;

$$I(L;C') = \frac{1}{n}\sum_{l}\sum_{c'} n_{l,c'} \log\left(\frac{n\, n_{l,c'}}{n_l\, n_{c'}}\right), \qquad I(L;C) = \frac{1}{n}\sum_{l}\sum_{c} n_{l,c} \log\left(\frac{n\, n_{l,c}}{n_l\, n_c}\right),$$

where $n_l = \sum_{c'} n_{l,c'}$.

$$
\begin{aligned}
& I(L;C') - I(L;C) \\
&= \frac{1}{n}\sum_{l}\sum_{c'} n_{l,c'} \log\!\left(\frac{n\, n_{l,c'}}{n_l\, n_{c'}}\right) - \frac{1}{n}\sum_{l}\sum_{c} n_{l,c} \log\!\left(\frac{n\, n_{l,c}}{n_l\, n_c}\right) \\
&= \frac{1}{n}\sum_{c}\sum_{c'\subseteq c}\sum_{l}\left[n_{l,c'} \log\!\left(\frac{n\, n_{l,c'}}{n_l\, n_{c'}}\right) - n_{l,c'} \log\!\left(\frac{n\, n_{l,c}}{n_l\, n_c}\right)\right] \\
&= \frac{1}{n}\sum_{c}\sum_{c'\subseteq c}\sum_{l} n_{l,c'} \log\!\left(\frac{n_{l,c'}\, n_c}{n_{l,c}\, n_{c'}}\right).
\end{aligned}
$$

Since every $c' \subseteq of c$, therefore $\frac{n_{l,c'}}{n_{c'}} \geq \frac{n_{l,c}}{n_c}$. Thus the value in the log is positive, and $I(L;C') \geq I(L;C)$ □

Lemma. 2. *Let C a community partition. Let C' be a refinement of C, i.e., $C' \preceq C$. Then $H(C') \geq H(C)$*

Proof. Let total size n. Based on the definition in Section 2

$$
H(C) \;=\; -\sum_{c}\frac{n_c}{n}\log\frac{n_c}{n}, \qquad H(C') \;=\; -\sum_{c'}\frac{n_{c'}}{n}\log\frac{n_{c'}}{n}.
$$

By grouping the c' under their parent c:

$$
\begin{aligned}
H(C') - H(C) &= -\sum_{c}\sum_{c'\subseteq c}\frac{n_{c'}}{n}\log\frac{n_{c'}}{n} \;+\; \sum_{c}\frac{n_c}{n}\log\frac{n_c}{n} \\
&= \frac{1}{n}\sum_{c}\left[-\sum_{c'\subseteq c} n_{c'}\log n_{c'} \;+\; n_c \log n_c\right].
\end{aligned}
$$

Since $f(x) = -x\log x$ is a concave function and $c' \subseteq c$, therefore,

$$
\sum_{c'\subseteq c} -\frac{n_{c'}}{n}\log\frac{n_{c'}}{n} \;\;\geq\;\; -\frac{n_c}{n}\log\frac{n_c}{n}.
$$

Thus, $H(C') \geq H(C)$.

□

Theorem 1. *Let L be labels; C a community partition. Let C' be a refinement of C, i.e., $C' \preceq C$. Then $NMI(C';L) > NMI(C;L)$ if and only if $\frac{\Delta I}{\Delta H} > \frac{NMI(C;L)}{2}$; where $\Delta I = I(C';L) - I(C;L)$ and $\Delta H = H(C';L) - H(C;L)$*

Proof.

$$
\mathrm{NMI}(C;L) \;=\; \frac{2\, I(C;L)}{H(C) + H(L)}.
$$

$$
I := I(C;L), \qquad I' := I(C';L), \qquad H := H(C), \qquad H' := H(C'), \qquad H_L := H(L).
$$

Also

$$
\Delta I := I' - I, \qquad \Delta H := H' - H.
$$

Based on Lemma 1 and 2 t $\Delta I \geq 0$ and $\Delta H \geq 0$. We do not consider edge case where $\Delta H = 0$. To show

$$\mathrm{NMI}(C';L) > \mathrm{NMI}(C;L) \quad \Longleftrightarrow \quad \frac{\Delta I}{\Delta H} > \frac{\mathrm{NMI}(C;L)}{2}.$$

$$\mathrm{NMI}(C';L) > \mathrm{NMI}(C;L) \iff \frac{2I'}{H'+H_L} > \frac{2I}{H+H_L}.$$

$$\frac{I'}{H'+H_L} > \frac{I}{H+H_L} \Longleftrightarrow I'(H+H_L) - I(H'+H_L) > 0.$$

Expand using $I' = I + \Delta I$ and $H' = H + \Delta H$:

$$(I+\Delta I)(H+H_L) - I(H+\Delta H+H_L) > 0.$$

Simplify terms (the $I(H+H_L)$ cancel):

$$\Delta I\,(H+H_L) - I\,\Delta H \;>\; 0.$$

Thus;

$$\Delta I\,(H+H_L) > I\,\Delta H \quad \Longleftrightarrow \quad \frac{\Delta I}{\Delta H} > \frac{I}{H+H_L}.$$

By definition $\mathrm{NMI}(C;L) = \dfrac{2I}{H+H_L}$, so $\dfrac{I}{H+H_L} = \dfrac{\mathrm{NMI}(C;L)}{2}$. Therefore

$$\mathrm{NMI}(C';L) > \mathrm{NMI}(C;L) \quad \Longleftrightarrow \quad \frac{\Delta I}{\Delta H} > \frac{\mathrm{NMI}(C;L)}{2}$$

□

6.2 Algorithms

Algorithm 1 Adaptive Resolution Search for Louvain

Require: Graph G, max gap $\Delta_{\max}$, `gap_range`$= [a, b]$
Ensure: `resolutions`, `community_list`

$\mathcal{C} \leftarrow \emptyset$; $Q \leftarrow \emptyset$
for $r \in \{0.5, 1.0\}$ **do** ▷ initial resolutions
  $(\mathcal{C}[r], Q[r]) \leftarrow \textsc{Louvain}(G, r)$
while true **do**
  $L \leftarrow \textsc{SortedKeys}(Q)$; $r_{\max} \leftarrow L[-1]$; $Q_{\max} \leftarrow Q[r_{\max}]$
  if $Q_{\max} \le \tau$ **then**
    break
  new_r $\leftarrow$ `None`
  for consecutive $(r_1, r_2) \in L$ **do**
    if $|Q[r_2] - Q[r_1]| > \Delta_{\max}$ **then**
      new_r $\leftarrow (r_1 + r_2)/2$; **break**
  if new_r$=$ `None` **then** ▷ extrapolate
    sample $\delta \sim \mathcal{U}[a, b]$; $Q^\star \leftarrow Q_{\max} - \delta$
    $s \leftarrow \textsc{EstimateSlope}(Q \text{ vs } r)$;
    new_r $\leftarrow r_{\max} + \dfrac{Q^\star - Q_{\max}}{s}$
  $(\mathcal{C}[\text{new\_r}], Q[\text{new\_r}]) \leftarrow \textsc{Louvain}(G, \text{new\_r})$
`resolutions` $\leftarrow \{\, r \in \textsc{SortedKeys}(Q) :\ Q[r] \ge \tau \,\}$
`community_list` $\leftarrow [\, \mathcal{C}[r] \ \text{ for } r \in$ `resolutions` $]$
return `resolutions`, `community_list`

Algorithm 2 Community-Augmented Feature Projection for Node Classification

Require: Graph $G = (V, E)$, node features $\mathbf{X} \in \mathbb{R}^{n \times d}$, resolution set $\Gamma = \{\gamma_1, \ldots, \gamma_T\}$, projection dimension d_c
Ensure: Predicted label distribution $\hat{\mathbf{Y}} \in \mathbb{R}^{n \times C}$
Initialize empty list of embeddings $\mathcal{E}_{\text{emb}} \leftarrow [\,]$
for $\gamma \in \Gamma$ **do**
    Compute community assignment $\mathbf{c}^{(\gamma)} \in \mathbb{N}^n$
    One-hot encode $\mathbf{c}^{(\gamma)}$: $\mathbf{H}^{(\gamma)} \in \{0,1\}^{n \times k_\gamma}$
    Project via trainable weights: $\mathbf{E}^{(\gamma)} \leftarrow \mathbf{H}^{(\gamma)}\mathbf{W}^{(\gamma)}$, where $\mathbf{W}^{(\gamma)} \in \mathbb{R}^{k_\gamma \times d_c}$
    Append $\mathbf{E}^{(\gamma)}$ to $\mathcal{E}_{\text{emb}}$
Concatenate all embeddings: $\mathbf{E} \leftarrow \text{Concat}(\mathcal{E}_{\text{emb}}) \in \mathbb{R}^{n \times (T \cdot d_c)}$
Concatenate with node features: $\mathbf{Z} \leftarrow \left[\mathbf{X} \,\|\, \mathbf{E}\right] \in \mathbb{R}^{n \times (d + T \cdot d_c)}$
Predict logits with MLP: $\mathbf{Y} \leftarrow f_\theta(\mathbf{Z}) \in \mathbb{R}^{n \times C}$
Apply softmax: $\hat{\mathbf{Y}} \leftarrow \text{softmax}(\mathbf{Y})$
return $\widehat{\mathbf{Y}}$

6.3 Hyperparameter Details

Dataset	$Q_{\min}$	ΔQ	Epochs	Batch	Hidden	Layers	Dropout	LR
Cora	0.1	0.2	200	128	256	3	0.5	1e-4
Pubmed	0.6	0.07	1000	4000	512	3	0.7	1e-4
Tolokers	0.3	0.1	1000	512	512	3	0.7	1e-4
Squirrel-filtered	0.4	0.1	300	128	512	0	0.5	1e-4
Chameleon-filtered	0.3	0.1	200	128	512	0	0.0	1e-4
Amazon-ratings	0.6	0.1	1500	512	512	3	0.5	1e-4
Actor	1.0	0.1	200	128	512	3	0.8	1e-4
Roman-empire	1.0	0.1	500	512	512	3	0.5	1e-4
Flickr	0.1	0.04	60	512	512	3	0.7	1e-4
Reddit	0.3	0.3	1000	8000	512	3	0.5	1e-4
Yelp	1.0	0.1	300	32000	2048	5	0.5	5e-5
AmazonProducts	1.0	0.1	200	64000	2048	5	0.5	5e-5
ogbn-products	0.3	0.1	400	32000	512	3	0.5	1e-4

Table 6: Training hyperparameters by dataset. $Q_{\min}$ is the *minimum modularity* threshold and ΔQ is the *maximum modularity gap*.

6.4 Cora: Accuracy vs. Minimum Modularity Threshold

Table 7: Cora: Cumulative (Q, Resolution, Communities) pairs included at each minimum modularity threshold $Q_{\min}$ (listed in run order), with accuracy. *Color coding:* pairs colored in blue are *newly added at that* $Q_{\min}$; pairs in gray were added at earlier thresholds and are carried over.

Min Modularity $Q_{\min}$	**Pairs** (Modularity, Resolution,Number of Communities)	**Accuracy**
1.0	—	76.61
0.9	—	76.61
0.8	(0.8526, 0.500, 90), (0.8120, 1.000, 103)	79.93
0.7	(0.8526, 0.500, 90), (0.8120, 1.000, 103), (0.7448, 2.606, 141)	83.66
0.6	(0.8526, 0.500, 90), (0.8120, 1.000, 103), (0.7448, 2.606, 141), (0.6841, 5.483, 170), (0.6006, 12.374, 298)	86.50
0.5	(0.8526, 0.500, 90), (0.8120, 1.000, 103), (0.7448, 2.606, 141), (0.6841, 5.483, 170), (0.6006, 12.374, 298), (0.5566, 20.068, 325)	84.55
0.4	(0.8526, 0.500, 90), (0.8120, 1.000, 103), (0.7448, 2.606, 141), (0.6841, 5.483, 170), (0.6006, 12.374, 298), (0.5566, 20.068, 325), (0.4909, 32.860, 373), (0.4231, 48.392, 430)	86.15
0.3	(0.8526, 0.500, 90), (0.8120, 1.000, 103), (0.7448, 2.606, 141), (0.6841, 5.483, 170), (0.6006, 12.374, 298), (0.5566, 20.068, 325), (0.4909, 32.860, 373), (0.3748, 63.924, 457), (0.4231, 48.392, 430)	85.26
0.2	(0.8526, 0.500, 90), (0.8120, 1.000, 103), (0.7448, 2.606, 141), (0.6841, 5.483, 170), (0.6006, 12.374, 298), (0.5566, 20.068, 325), (0.4909, 32.860, 373), (0.3748, 63.924, 457), (0.4231, 48.392, 430), (0.2784, 95.726, 541)	82.59
0.1	(0.8526, 0.500, 90), (0.8120, 1.000, 103), (0.7448, 2.606, 141), (0.6841, 5.483, 170), (0.6006, 12.374, 298), (0.5566, 20.068, 325), (0.4909, 32.860, 373), (0.3748, 63.924, 457), (0.4231, 48.392, 430), (0.2784, 95.726, 541), (0.1792, 136.430, 672)	88.10
0.0	(0.8526, 0.500, 90), (0.8120, 1.000, 103), (0.7448, 2.606, 141), (0.6841, 5.483, 170), (0.6006, 12.374, 298), (0.5566, 20.068, 325), (0.4909, 32.860, 373), (0.3748, 63.924, 457), (0.4231, 48.392, 430), (0.2784, 95.726, 541), (0.1792, 136.430, 672), (0.0958, 175.819, 768)	86.32

6.5 Models for homophilic graphs

GCN. Applies a linear map followed by aggregation with the symmetrically normalized adjacency (after adding self-loops), corresponding to a first-order spectral/Chebyshev approximation (Kipf & Welling, 2017).

GAT. Learns attention coefficients over neighbors via masked self-attention and aggregates them with a softmax-weighted sum, enabling data-dependent receptive fields (Veličković et al., 2018).

GraphSAGE. Performs permutation-invariant neighbor aggregation (e.g., mean, max-pooling, LSTM) with fixed fan-out sampling per layer for scalable, inductive mini-batch training on large graphs (Hamilton et al., 2017).

6.6 Models for heterophilic graphs

H_2GCN. Separates ego and neighbor embeddings, aggregates higher-order neighborhoods, and combines intermediate representations to improve robustness under heterophily (Zhu et al., 2020a).

LinkX. Separately embeds node features and adjacency (structural) information with MLPs and concatenates them, capturing complementary attribute and topology signals that scale to non-homophilous graphs (Lim et al., 2021).

GPR-GNN. Learns signed polynomial (Generalized PageRank) propagation weights, adapting the filter to both homophilous and heterophilous label patterns and mitigating over-smoothing (Chien et al., 2021).

FSGNN. Applies soft selection over hop-wise aggregated features with "hop-normalization," effectively decoupling aggregation depth from message passing for a simple, shallow baseline that performs well under heterophily (**?**).

GloGNN. Augments propagation with learnable correlations to global nodes (including signed coefficients), enabling long-range information flow and improved grouping on heterophilous graphs (Li et al., 2022).

FAGCN. Uses a self-gating, frequency-adaptive mechanism to balance low- and high-frequency components during message passing, improving robustness across homophily regimes (Bo et al., 2021).

GBK-GNN. Employs bi-kernel feature transformations with a gating mechanism to integrate homophily- and heterophily-sensitive signals within a single architecture (Du et al., 2022).

JacobiConv. Adopts an orthogonal Jacobi-polynomial spectral basis (often without non-linearities) to learn flexible filters suited to varying graph signal densities, yielding strong performance on heterophilous data (Wang & Zhang, 2022).

6.7 Sampling methods for scalable GNNs

GraphSAGE (node/neighbor sampling). Samples a fixed fan-out of neighbors per layer and learns permutation-invariant aggregators, limiting the receptive field and enabling inductive, mini-batch training on large graphs (Hamilton et al., 2017).

FastGCN (layer-wise node sampling). Recasts graph convolution as an expectation over nodes and draws i.i.d. node sets at each layer via importance sampling, decoupling batch size from degree and reducing estimator variance (Chen et al., 2018b).

S-GCN / VR-GCN (layer-wise with control variates). Introduces control-variates using historical activations to stabilize gradients under small per-layer samples and achieve faster, provable convergence to the full-batch optimum (Chen et al., 2018a).

ClusterGCN (subgraph/block sampling). Partitions the graph and samples dense clusters as mini-batches, restricting propagation within blocks to boost edge coverage, cache locality, and memory efficiency at scale (Chiang et al., 2019).

GraphSAINT (subgraph sampling with bias correction). Constructs mini-batches by sampling subgraphs (node/edge/random-walk policies) and applies unbiased normalization to correct sampling bias, yielding strong accuracy–efficiency trade-offs on large graphs (**?**).

6.8 Datasets

We evaluate on two groups of benchmarks that stress complementary regimes.

Large-scale graphs. We use Flickr, Reddit, Yelp, AmazonProducts, and ogbn-products. Flickr/Yelp/AmazonProducts come from GraphSAINT; Reddit from GraphSAGE; ogbn-products from OGB (**?**Hamilton et al., 2017; Hu et al., 2020). Table 8 reports sizes, features, classes, and splits.

Homophilous and heterophilous graphs. We include Cora, PubMed, Actor, Chameleon-filtered, Squirrel-filtered, Amazon-ratings, Tolokers, and Roman-empire. For the filtered Wikipedia, Roman-empire, Amazon-ratings, and Tolokers datasets, we use the exact settings and splits of Platonov et al. (2023); Cora, PubMed, and Actor follow standard preprocessing (Sen et al., 2008; Pei et al., 2020; Lim et al., 2021). Table 9 lists summary stats, edge homophily h_e, and metrics.

Table 8: Dataset statistics ("m" stands for **m**ulti-class classification, and "s" for **s**ingle-class.)

Dataset	Nodes	Edges	Avg. Degree	Feature	Classes	Train / Val / Test
Flickr	89,250	899,756	10	500	7 (s)	0.50 / 0.25 / 0.25
Reddit	232,965	11,606,919	50	602	41 (s)	0.66 / 0.10 / 0.24
Yelp	716,847	6,977,410	10	300	100 (m)	0.75 / 0.10 / 0.15
AmazonProducts	1,598,960	132,169,734	83	200	107 (m)	0.85 / 0.05 / 0.10
ogbn-products	2,449,029	61,859,140	50.5	100	47 (s)	0.08 / 0.02 / 0.90

Table 9: Dataset statistics with edge homophily h_e and evaluation metric ("Acc" for Accuracy, "ROC-AUC" for Area Under ROC).

Dataset	Nodes	Edges	Avg. Degree	Feature	Classes	Train / Val / Test	h_e	Metric
Cora	2,708	5,429	4	1,433	7 (s)	0.60 / 0.20 / 0.20	0.810	Acc
PubMed	19,717	44,324	5	500	3 (s)	0.60 / 0.20 / 0.20	0.802	Acc
Actor	7,600	30,019	8	932	5 (s)	0.60 / 0.20 / 0.20	0.216	Acc
Squirrel-filtered	2,223	65,718	59	2,089	5 (s)	0.50 / 0.25 / 0.25	0.207	Acc
Chameleon-filtered	890	13,584	31	2,325	5 (s)	0.50 / 0.25 / 0.25	0.236	Acc
Amazon-ratings	24,492	93,050	8	300	5 (s)	0.50 / 0.25 / 0.25	0.380	Acc
Tolokers	11,758	519,000	88	10	2 (s)	0.50 / 0.25 / 0.25	0.595	ROC-AUC
Roman-empire	22,662	32,927	3	300	18 (s)	0.50 / 0.25 / 0.25	0.047	Acc