DEEP RECYCLE: Beyond GNNs for Large-Scale Community Detection

Caleb Fernandes
Department of Computer Science
University of South Florida
ctkcl@jmu.edu

Audrey Knight
Department of Mathematics
Towson University
ynhcf8@jmu.edu

Samuel Stoke
Department of Electrical and Computer Engineering
Virginia Tech
samuelstoke@vt.edu

Behnaz Moradi-Jamei
Department of Mathematics and Statistics
James Madison University
moradibx@jmu.edu

Abstract

Graph Neural Networks (GNNs) typically aggregate local neighborhood information without explicitly capturing higher-order structural features, such as cycles, limiting their effectiveness in unsupervised community detection tasks. We introduce ReCYCLE Net, a novel cycle-aware GNN framework leveraging Renewal Non-Backtracking Random Walks (RNBRW) to quantify cycle participation, significantly enhancing inductive biases for community detection. RNBRW weights are integrated into Graph Attention Networks (GAT) as attention biases and loss coefficients, improving structural coherence and community boundary detection. Experiments conducted on benchmark datasets (Karate Club, Political Books, Cora, DBLP, Amazon) demonstrate substantial improvements over baselines in modularity, normalized mutual information (NMI) and adjusted Rand index (ARI).

1 Introduction

Deep ReCYCLE: Learning to Reason with Cycles for Community Detection

In the interconnected world around us, from social media networks to biological systems, communities naturally form through complex patterns of relationships. However, *one hammer doesn't fit all* in community detection. Different domains exhibit structurally distinct mechanisms for community formation that require domain-specific approaches rather than generic, one-size-fits-all solutions.

In meaningful cycle-rich domains such as social networks, biological interaction graphs, and knowledge graphs, one of the most fundamental yet overlooked patterns is the presence of **cycles**: friendship triangles that indicate trust, feedback loops that represent functional pathways, and citation circles that reveal research communities. These aren't just mathematical curiosities but essential building blocks with semantic meaning that reveal how communities actually form and function in these specific contexts.

Traditional community detection methods apply universal algorithms that treat all connections equally across domains, missing the critical insight that community detection methods should be tailored to domain-specific structural patterns. More importantly, in meaningful cycle-rich graphs is essential for understanding how these networks actually organize themselves.

While classical algorithms like Louvain are fast, they remain domain-agnostic and ignore crucial structural patterns that carry semantic meaning in specific contexts. Modern machine learning approaches using Graph Neural Networks (GNNs) can learn from data, but typically apply generic message-passing schemes that diffuse information in ways that wash out the very domain-specific cycle patterns that encode meaningful community signals.

Our research introduces **Deep ReCYCLE**, a novel framework that recognizes the need for domain-tailored community detection by putting meaningful cycle reasoning at the center of the learning process. We utilize a method called *Renewal Non-Backtracking Random Walks (RNBRW)* that efficiently quantifies how much each connection participates in meaningful cycles, without the computational burden of enumerating every cycle in the network [MJSPC+21].

This cycle-awareness is integrated into neural networks through two key innovations:

1. Teaching the network to pay more attention to connections that participate in meaningful cycles when aggregating information
2. Training the network with objectives that explicitly reward cycle-based community structure—both tailored specifically for domains where cycles encode semantically meaningful community signals

We tested our approach on networks ranging from the classic 34-node Karate Club to the massive 2.4-million-node Amazon product networks. The results demonstrate that domain-tailored, meaningful cycle-aware learning consistently outperforms generic methods, achieving accuracy scores of 0.88 on benchmark datasets. Our ablation studies confirm that both components of cycle reasoning are necessary for optimal performance.

This work opens new possibilities for understanding complex networks by recognizing that effective community detection requires domain-specific reasoning. In meaningful cycle-rich domains, the patterns we often overlook (the cycles with semantic meaning) may be the very patterns that matter most for revealing hidden community structures in everything from drug discovery networks to social recommendation systems.

2 Background

Graph Neural Networks (GNNs) have revolutionized learning on graph-structured data, particularly in tasks like community detection, where capturing cohesive subgraphs is essential. Traditional GNNs, such as Graph Convolutional Networks (GCNs) [KW17], perform semi-supervised classification by aggregating node features through localized convolutions based on the graph Laplacian. However, GCNs suffer from limitations including over-smoothing, sensitivity to graph noise, and inability to handle inductive settings or variable-sized inputs. As analyzed by Li et al. [LHW18], repeated convolutions in GCNs equate to Laplacian smoothing, which can collapse features and degrade performance in semi-supervised learning. To address label scarcity, they propose co-training with partially absorbing random walks and self-training via confident predictions, enhancing robustness.

Attention mechanisms have been introduced to mitigate these issues. Veličković et al. [VCC+18] propose Graph Attention Networks (GATs), which learn dynamic weights for neighbor messages via a masked attention mechanism. This allows GATs to prioritize important connections, enabling parallelism, inductive generalization, and noise resistance. However, standard GATs overlook edge features and rely heavily on node attributes, necessitating engineered features for effective attention. Wang et al. [WYHL19] extend GAT with Constrained GAT (C-GAT), incorporating large-margin constraints on attention coefficients to preserve class boundaries and structural regularization to combat over-fitting, theoretically reducing over-smoothing.

For inductive and scalable learning on large graphs, Hamilton et al. [HYL18] introduce GraphSAGE, which aggregates features from sampled neighborhoods using mean, long short-term memory (LSTM), or pooling functions. Unlike transductive methods, GraphSAGE generalizes to unseen nodes, outperforming baselines like DeepWalk on diverse datasets. Zeng et al. [ZZS+20] further scale GNNs with GraphSAINT, sampling subgraphs (node/edge/random walk) to avoid neighbor explosion, achieving high accuracy on massive graphs like Amazon. Chiang et al. [CLS+19] propose Cluster-GCN, clustering nodes (e.g., via METIS) and training on cluster unions for efficiency, scaling to Amazon2M while reducing memory.

Robustness remains a challenge, as GNNs are vulnerable to adversarial perturbations. Jin et al. [JML+20] present Pro-GNN, which jointly optimizes for a clean graph structure (via sparsity, low-rank, and smoothness priors) and GNN parameters, effectively defending against attacks. This is particularly relevant for real-world applications, though assumptions like sparsity may not hold for dense graphs.

In community detection, foundational surveys like Fortunato [FH16] emphasize probabilistic models over edge-frequency methods, advocating null models (e.g., random graphs) for significance testing. Recent work by Moradi-Jamei et al. [MJKCK21] demonstrates that Renewal Non-Backtracking Random Walks (RNBRW) enhance community detection when coupled with Louvain, by pre-weighting edges based on cyclic structures to improve partitioning quality in sparse graphs. Our work builds directly on RNBRW by extending its cycle quantification into unsupervised GNN frameworks, infusing it across features, aggregation, and losses to address gaps in handling higher-order motifs for scalable detection.

Literature Review [KW17]

[VCC+18] Veličković proposes a GNN architecture that weights the importance of messages passed to the target node, dependent on the attention coefficient between two nodes. Traditional approaches to GNNs like GCNs have issues as they are unable to solve inductive problems (generalize embeddings to unseen graphs), weight all messages by neighbors equally, cannot adapt to variable-sized inputs, exhibit over-smoothing, and sensitivity to graph noise. Veličković solves these by implementing a masked attention mechanism, a learnable neural network node layer that outputs an attention coefficient which weights the importance of incoming messages from neighboring nodes. This Graph Attention Networks (GAT) are parallelizable and inductive, making them very efficient. GAT's weight ensures low sensitivity to noise, which can be beneficial for structural analysis. For example, RNBRWs' weights can be used in conjunction with attention coefficients in the architecture to create better attention coefficients. The downside is that traditional GATs do not use edge features, rather they rely heavily on node features. To incorporate edge features would require a variation of GAT, and node features have to be engineered for the attention mechanism to provide insightful coefficients.

[JML+20] Jin et al. address the vulnerability of graph neural networks (GNNs) to adversarial attacks. Small perturbations to graph structure or node features, for example adding or removing edges, often significantly degrade model performance. Given that GNNs rely heavily on graph topology for message passing, such attacks pose a serious threat, particularly in high-stakes domains such as banking or threat detection. To mitigate this, the authors propose Pro-GNN, a robust training framework that explicitly models and removes adversarial noise from the graph before learning. The approach is grounded in structural assumptions common to real-world graphs—namely, sparsity, low-rank structure, and feature smoothness. Pro-GNN also formulates a joint optimization problem to recover a clean adjacency matrix and train GNN parameters simultaneously. The objective function includes three main components, the first being L1 and nuclear norm regularization to promote sparsity and low rank in the estimated graph, helping eliminate perturbed edges. Second, a smoothness penalty based on the graph Laplacian to ensure neighboring nodes retain similar features is used. Third a task-specific loss that preserves prediction performance, controlled by a tunable hyperparameter is proposed. Pro-GNN demonstrates strong resilience to structural attacks, making it particularly promising for financial applications, where data integrity and resistance to tampering are critical. However, the limitation to this model is the general assumptions it places on graphs. This model would not be beneficial to use on a graph that is meant to be dense, like a social network.

[WYHL19] Wang et al. (2019) extend the Graph Attention Network (GAT) framework by introducing Constrained Graph Attention Networks (C-GAT), which aim to mitigate two key limitations of standard GATs: over-smoothing and over-fitting. While GATs excel at learning node representations by weighting neighbor messages through attention, their flexibility can lead to attention weights that do not respect class boundaries, causing feature collapse near decision boundaries. To address this, the authors incorporate large-margin constraints on attention coefficients, ensuring that nodes belonging to the same class exert notably higher attention on each other compared to nodes from different classes. Additionally, they propose structural regularization to curb over-parameterization and prevent over-fitting. Theoretical analysis confirms that these margin-based constraints effectively reduce over-smoothing by maintaining class separability, while empirical results across multiple benchmarks, both transductive and inductive, demonstrate that C-GAT consistently outperforms standard GAT.

[HYL18] In the paper "Inductive Representation Learning on Large Graphs (GraphSAGE)" by Hamilton et al. (2017), the authors introduce the GraphSAGE graph neural network (GNN) model, testing it on multiple datasets against existing baseline methods. The GraphSAGE model leverages node features, rather than relying solely on matrix factorization, to learn node embeddings. This is done by iteratively aggregating features from sampled local neighborhoods of nodes. The loss function encourages similar nodes to have close embeddings and differing nodes to have distant embeddings. GraphSAGE supports both unsupervised and supervised learning tasks. Notably, the inductive nature of the GraphSAGE approach allows it to generalize embeddings to unseen nodes, addressing limitations of transductive methods that struggle with large or evolving graphs. Three distinct aggregation functions—mean, LSTM-based, and pooling—are evaluated against four baselines: a random classifier, logistic regression feature-based classifier, DeepWalk, and DeepWalk combined with node features. The models are assessed on three datasets: Web of Science citation data, Reddit posts categorized by community, and protein-protein interaction (PPI) networks. GraphSAGE significantly outperforms all baseline methods across these datasets. Future work suggested by the authors includes utilizing non-uniform neighborhood sampling methods or incorporating additional structural graph information.

[ZZS+20] In GraphSAINT (Graph SAmpling based INductive learning meThod) (2019), Zeng et al. introduce a graph neural network model optimized for node classification tasks. The GraphSAINT approach involves sampling subgraphs from the original training graph, constructing graph convolutional networks (GCNs) for each subgraph, and propagating these GCNs through forward and backward passes. The subgraphs are carefully sampled to reduce variance and prevent bias towards frequently appearing ("popular") nodes. Multiple sampling methods are explored, including node sampling, edge sampling, random walk sampling, and multi-dimensional random walk sampling. These sampling strategies require minimal preprocessing computation on the training graph. GraphSAINT employs an inductive approach and effectively avoids the "neighbor explosion" problem, where node neighborhoods grow exponentially with increasing layers. The GraphSAINT model, tested with each sampling method independently, is evaluated across five diverse datasets including protein-protein interactions (PPI), online image categorization (Flickr), community prediction from Reddit comments, business categorization based on Yelp user interactions, and product categorization from Amazon reviews. GraphSAINT consistently demonstrates higher accuracy compared to baseline methods (including GCN, GraphSAGE, FastGCN, S-GCN, AS-GCN, and Cluster-GCN). Additionally, GraphSAINT exhibits greater scalability and reduced training complexity due to its graph sampling methodology, inductive capability, avoidance of neighbor explosion, and efficient preprocessing.

[CLS+19] In the paper "Cluster-GCN" by Chiang et al. (2019), the authors address critical limitations of existing GNN models by introducing the Cluster-GCN model. Previous models typically encounter issues with memory usage, computational time per epoch, or convergence. Cluster-GCN resolves these challenges by sampling subgraphs created through clustering the nodes into groups using algorithms like METIS, and then selecting random unions of these clusters for each training iteration. Graph convolutional networks (GCNs) are computed on these cluster unions, and their outputs are aggregated during forward and backward propagation. The Cluster-GCN approach significantly improves training efficiency by minimizing cross-cluster communication overhead. The model is evaluated against existing methods on several datasets, including protein-protein interaction (PPI), Reddit, Amazon, and Amazon2M—a new, substantially larger dataset introduced for scalability evaluation. Cluster-GCN demonstrates improved accuracy as well as significantly enhanced scalability in both time and memory efficiency compared to existing approaches.

[FH16] Community detection in networks: A user guide This article provides a comprehensive introduction to the problem of community detection in networks. It outlines foundational concepts and traces the evolution of approaches from classical to modern techniques. Traditional methods typically rely on edge frequency to identify communities, while newer approaches leverage more sophisticated models that account for the probabilistic nature of connections, better capturing the inherent complexity of real world networks. The paper explores various definitions of what constitutes a community, ultimately arguing that no single universal definition is necessary. Instead, it emphasizes the importance of defining clusters clearly in advance to ensure meaningful result validation. The authors also advocate for using random graphs as null models when testing community significance. The article then surveys a range of detection methods, including consensus clustering, spectral techniques, statistical inference, and optimization. A key takeaway is the need to assess the statistical significance of any detected communities.

[LHW18] Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning This paper explores the foundational principles of graph convolutional neural networks (GCNs) and examines their limitations, particularly in semi-supervised settings. Li et al. reveals a key theoretical insight by connecting graph convolutions to Laplacian smoothing, highlighting how repeated convolution operations can over-smooth features. To mitigate GCNs' dependency on large amounts of labeled data, the authors propose two strategies, co-training and self-training. In the co-training approach, a partially absorbing random walk model is used alongside the GCN to augment the learning process and reduce the reliance on labeled validation data. For self-training, the model iteratively adds its most confident predictions to the labeled set, effectively expanding the training data. This combined strategy significantly enhances performance, especially in scenarios where labeled data is scarce.

[KW17] Semi-Supervised Classification with Graph Convolutional Networks This paper presents a scalable semi-supervised learning method tailored for graph structured data, based on an efficient variant of convolutional neural networks. The proposed approach generates hidden layer representations that incorporate both the graph topology and node features, an aspect particularly relevant to cycle aware reasoning. Evaluated on citation network datasets, the model outperforms several baseline methods. A key contribution is the introduction of a layer-wise propagation rule for neural network models, which enables effective and scalable learning on graphs. However, the method comes with some limitations. For example, it has relatively high memory requirements, does not support directed graphs, and assumes both locality and equal importance of self-loops and neighboring connections. These assumptions that may not hold in all real-world scenarios, so understanding the application is important.

2.1 Related Methods

2.1.1 Community Detection and RNBRW

Classical community detection methods such as Louvain algorithm effectively identify modular structures but ignore specific structural features like cycles that characterize cohesive subgraphs [BGLL08]. RNBRW addresses this by quantifying edge participation in cyclic structures through non-backtracking random walks [MJSPC$^+$21].

2.1.2 Graph Neural Networks

GNNs leverage graph topology and node features to generate meaningful embeddings. Popular variants like GCN and GAT use distinct aggregation schemes; GCN applies neighborhood averaging [KW17], while GAT utilizes attention mechanisms to weight neighbors dynamically [VCC$^+$18]. Both methods, however, fail to exploit higher-order cyclic structural information explicitly.

2.1.3 GAT

Graph Attention Networks (GATs), introduced by Veličković et al., use attention mechanisms to assign dynamic weights to neighbor nodes, enabling scalability, inductive generalization, and improved noise resistance. While they support parallel computation, standard GATs rely heavily on node features and do not incorporate edge information or higher-order structure. To address this, Wang et al. propose Constrained GAT (C-GAT), which adds margin-based constraints and structural regularization to preserve class boundaries and mitigate over-smoothing. These enhancements improve GAT's ability to capture structural patterns while maintaining scalability.

2.1.4 DGI

While GNN training has been widely optimized, scalable and efficient inference remains a challenge due to neighbor explosion during evaluation. Layer-wise inference improves scalability by using only one-hop neighbors per layer but requires manual model decomposition and memory management. Deep Graph Inference (DGI) addresses this by automatically converting training code into layer-wise execution, supporting large graphs and preserving structural information. Experiments show that DGI significantly improves inference speed—up to 1,000×—across models and datasets [YYZ+22].

2.1.5 GRACE

GRACE is a tool that enables consistent implementation of common compressed communication methods. GRACE is useful to evaluate many deep neural networks paired with various datasets and configurations. This work addresses fundamental assumptions about experiments, such as assuming convexity and not considering the cost of compression and decompression [XHA+21].

2.1.6 DAEGC

DAEGC is a method that addresses performance issues due to non-goal-directed graph embeddings. This method aims to create application specific deep learning approach. An attention network is used to encode topological structure and node content. An inner product decoder is trained to recover the topological structure and node content. Additionally self-training is used to generate soft labels which can supervise the embedding updating. DAEGC achieved state of the art results compared to contemporary methods [WPH+19].

2.1.7 BGRL

Bootstrapped graph latents (BGRL) is a graph representation learning method. BGRL predicts alternative augmentations of the input. BGRL uses simple augmentation and eliminates the requirement of negative examples. BGRL acheives state of the art results and reduces memory cost. This method is scalable especially when using semi-supervised learning [TTA+23].

2.1.8 SDCN

Structural Deep Clustering Networks (SDCN) are designed to incorporate graph topology into the clustering process. The approach introduces a delivery operator that passes representations from an autoencoder to a corresponding GCN layer. To enhance this interaction, the method also employs self-supervision to jointly optimize the autoencoder and delivery mechanism. This framework achieves state-of-the-art results in clustering tasks [BWS+20].

2.1.9 DMON

Node pooling methods perform compression but often fail to capture the underlying structure of a network. To investigate this limitation, researchers experimented with varying the signal-to-noise ratio in both edges and features. To address this challenge, they proposed deep modularity networks (DMON) an unsupervised pooling approach designed to better preserve graph structure during compression. DMON outperforms baseline methods in capturing meaningful communities [TPPM23].

3 Methodology

Feature Degree: if we use degree as a feature for GAT, then two nodes i,j that are neighbors and both have similar degrees will be weighted higher. As such, GAT would emphasize bridges between communities since this weight is high, RNBRW can make this better because a bridge in RNBRW would be weighted as 0 since it does not complete a cycle. So this is where it would increase community detection.

Our proposed RNBRW-GNN framework integrates Renewal Non-Backtracking Random Walks (RNBRW) into the message passing and loss function of graph neural networks to reinforce community structure. We describe the RNBRW-based edge weighting, the modified message passing scheme, and the construction of unsupervised objectives.

3.1 RNBRW Edge Weighting

Let $G = (V, E)$ be an undirected graph with node set V and edge set E. RNBRW computes edge importance by counting how frequently edges appear in completed cycles during renewal non-backtracking walks. Specifically, we define the edge weight:

$$w_{ij} = \frac{1}{T} \sum_{t=1}^{T} \mathbb{I}[(i,j) \in \mathcal{C}_t], \tag{1}$$

where $\mathcal{C}_t$ denotes the set of edges forming a completed cycle in the t-th RNBRW, and T is the number of walks. These weights define a topology-aware adjacency matrix $\widetilde{A} = [w_{ij}]$.

3.2 Node Feature Augmentation with RNBRW

For each node v_i, we compute the average RNBRW edge weight among its incident edges:

$$f_i^{\text{RNBRW}} = \frac{1}{\deg(i)} \sum_{j \in \mathcal{N}(i)} w_{ij}^{\text{RNBRW}}$$

This scalar is concatenated to the input feature vector, extending the representation space to reflect cyclic importance:

$$x_i^{\text{aug}} = [x_i; f_i^{\text{RNBRW}}]$$

3.3 Cycle-Aware GAT Aggregation

We introduce a modified Graph Attention Network (GAT) architecture that explicitly incorporates local cycle structure into the message-passing process, referred to as Cycle-Aware GAT (CA-GAT). Traditional GAT computes attention coefficients based solely on node features, without considering the underlying topological context. However, cycles are fundamental components of graph structure, particularly in community-rich networks, and can offer meaningful constraints on information flow.

To capture this, we enhance the GAT aggregation mechanism by biasing the attention weights using a cycle-based structural prior. Specifically, we compute Renewal Non-Backtracking Random Walk (RNBRW) edge weights that quantify the relative importance of each edge in preserving cycle connectivity. These weights are then injected into the attention mechanism to influence the learned coefficients.

Formally, given a graph $G = (V, E)$ with node features $h_i \in \mathbb{R}^F$, we compute the standard GAT attention score between nodes i and j as:

$$e_{ij} = \text{LeakyReLU}(a^\top [W h_i \parallel W h_j]),$$

where W is a learnable weight matrix and a is the attention vector. In CA-GAT, we introduce a multiplicative bias using the normalized RNBRW score r_{ij}:

$$\tilde{e}_{ij} = r_{ij} \cdot e_{ij},$$

followed by the usual softmax normalization across the neighborhood $\mathcal{N}(i)$:

$$\alpha_{ij} = \frac{\exp(\tilde{e}_{ij})}{\sum_{k \in \mathcal{N}(i)} \exp(\tilde{e}_{ik})}.$$

This formulation allows the network to prioritize messages along edges that are more cycle-relevant, encouraging embeddings that are consistent with the graph's cyclic structure. The final node update rule remains:

$$h_i' = \sigma \left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij} W h_j \right),$$

where σ is a non-linear activation function such as ELU.

By incorporating RNBRW-based cycle awareness, CA-GAT can better capture community boundaries and structural coherence, particularly in networks where cycles delineate functional subgraphs.

3.4 Cycle-Aware GCN Aggregation

In a standard GCN, the layer-wise propagation rule uses the normalized adjacency matrix:

$$A_{\text{norm}} = \widetilde{D}^{-1/2}(A+I)\widetilde{D}^{-1/2}$$

We instead build a weighted adjacency $\widetilde{A}$ using the RNBRW edge scores:

$$\widetilde{A}_{ij} = \log(w_{ij}^{\text{RNBRW}} + \epsilon)$$

where ϵ is a small constant to prevent undefined values. This matrix is then normalized as above and used for message passing.

3.5 RNBRW-GNN Layer

Given the edge-weighted matrix $\widetilde{A}$ and degree matrix $\widetilde{D}$, we define a message passing layer:

$$H^{(l+1)} = \sigma(\widetilde{D}^{-1/2}\widetilde{A}\widetilde{D}^{-1/2}H^{(l)}W^{(l)}), \tag{2}$$

where $H^{(l)}$ is the node embedding at layer l, $W^{(l)}$ is a learnable weight matrix, and σ is an activation function. This layer emphasizes message propagation over structurally cohesive paths identified by RNBRW.

3.6 Unsupervised Learning Objectives

We combine multiple unsupervised losses:

Modularity Loss. We define a soft cluster assignment Q from node embeddings and maximize modularity:

$$\mathcal{L}\text{mod} = -\frac{1}{2m}\sum i,j \left(w_{ij} - \frac{d_i d_j}{2m}\right)(Q_i^\top Q_j), \tag{3}$$

where d_i is the degree of node i in $\widetilde{A}$ and $m = \frac{1}{2}\sum_{ij} w_{ij}$.

Laplacian Smoothness. The embeddings are encouraged to be smooth over structurally relevant edges:

$$\mathcal{L}_{\text{lap}} = \text{Tr}(H^\top(I - \widetilde{D}^{-1/2}\widetilde{A}\widetilde{D}^{-1/2})H). \tag{4}$$

Contrastive Loss. Following GRACE, we use RNBRW-based sampling to define positives and train a contrastive encoder f:

$$\mathcal{L}\text{contrastive} = -\log\frac{\exp(\text{sim}(f(x), f(x^+))/\tau)}{\sum x^- \exp(\text{sim}(f(x), f(x^-))/\tau)}, \tag{5}$$

where x^+ is a positive sample from the RNBRW neighborhood and x^- are negatives.

The *total loss* is a weighted combination:

$$\mathcal{L} = \lambda_1\mathcal{L}_{\text{mod}} + \lambda_2\mathcal{L}_{\text{lap}} + \lambda_3\mathcal{L}_{\text{contrastive}}. \tag{6}$$

3.7 GraphSAINT Methodology

We next describe our implementation of the GraphSAINT framework, including both the standard baseline and our RNBRW-enhanced variants.

3.7.1 Baseline GraphSAINT

We implement the standard GraphSAINT random-walk sampler [ZZS+20] and a two-layer GCN encoder to extract node embeddings for community detection. Given a graph $G = (V, E)$ with features $X \in \mathbb{R}^{n \times d}$, we sample subgraphs via

$$\text{sampler} = \text{GraphSAINTRandomWalkSampler}(G, ; \text{batch\_size} = 16, ; \text{walk\_length} = 3, ; \text{num\_steps} = 10). \tag{7}$$

Over each sampled subgraph we train a two-layer GCN:

$$H^{(1)} = \sigma\big(\text{GCNConv}(X, E)\big), \quad H^{(2)} = \text{GCNConv}\big(H^{(1)}, E\big), \tag{8}$$

for 300 epochs using Adam with learning rate 0.01. Final embeddings $H^{(2)} \in \mathbb{R}^{n \times 16}$ are clustered via k-means and evaluated by ARI and NMI.

3.7.2 RNBRW-Enhanced Variants

We inject cycle information from Renewal Non-Backtracking Random Walks (RNBRW) in two ways:

1. **Node-Feature Augmentation.** Compute RNBRW edge weights w_{ij} via $2|E|$ non-backtracking walks. Aggregate per node:

$$f_i = \frac{1}{\deg(i)} \sum_{(i,j) \in E} w_{ij}, \quad X \leftarrow [X; |; f], \tag{9}$$

   then repeat the baseline pipeline with augmented features.

2. **Weighted Sampling (In Progress).** Modify the sampler so that, at each step, a neighbor j of node i is chosen with probability proportional to w_{ij}. This requires a custom sampler or use of GraphSAINT's edge-importance sampler.

3.7.3 Full Pipeline Pseudocode

For clarity, the complete pipeline in PyTorch-like pseudocode is shown below:

```
1) Optional RNBRW node feature augmentation

if use_rnbrw:
rnbrw_w = compute_rnbrw(edge_index, T=2 * num_edges)
f  = aggregate_per_node(edge_index, rnbrw_w)
X  = torch.cat([X, f.unsqueeze(1)], dim=1)

2) Initialize GraphSAINT sampler

sampler = GraphSAINTRandomWalkSampler(
data,
batch_size=16,
walk_length=3,
num_steps=10
)

3) GCN training loop

model     = GCN(in_feats=X.size(1), hid_feats=32, out_feats=16)
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
for epoch in range(300):
for subdata in sampler:
optimizer.zero_grad()
H_sub = model(subdata.x, subdata.edge_index)
loss  = unsup_loss(H_sub)  # e.g., contrastive or reconstruction
loss.backward()
optimizer.step()

4) Extract embeddings & cluster
```

```
H_full = model(data.x, data.edge_index).detach().cpu().numpy()
preds  = KMeans(n_clusters=K).fit_predict(H_full)
```

3.8 Toy Counterexample: Beyond 1-WL with RNBRW

To demonstrate RNBRW-GNN's expressive advantage over 1-WL GNNs, we construct a pair of 6-node 3-regular graphs:

Graph A (Triangular Prism): Two triangles $1, 2, 3$ and $4, 5, 6$ connected in parallel:

- Edges: $(1, 2), (2, 3), (3, 1), (4, 5), (5, 6), (6, 4), (1, 4), (2, 5), (3, 6)$
- Contains 2 triangles and 3 distinct 4-cycles

Graph B (Chordal Cycle): A 6-cycle with chords:

- Edges: $(1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 1), (1, 4), (2, 5), (3, 6)$
- Contains no 3-cycles and only one 6-cycle

These graphs are indistinguishable by 1-WL due to matching degree profiles and local neighborhoods. However, RNBRW distinguishes them via the presence and frequency of short cycles, which bias edge weights and influence message passing.

We simulate RNBRW on both and visualize edge weights: Graph A shows high weight concentration on 3-cycle edges, while Graph B distributes weights evenly. This leads RNBRW-GNN to separate the two graphs in embedding space, confirmed by t-SNE plots and cosine heatmaps. GCN embeddings, in contrast, collapse due to symmetry.

This example illustrates that RNBRW-GNN can distinguish non-isomorphic graphs with identical 1-WL signatures by exploiting global recurrence patterns.

3.9 Toy Counterexample: Beyond 1-WL with RNBRW

To demonstrate RNBRW-GNN's expressive advantage over 1-WL GNNs, we construct a pair of 6-node 3-regular graphs:

Graph A (Triangular Prism): Two triangles $1, 2, 3$ and $4, 5, 6$ connected in parallel:

- Edges: $(1, 2), (2, 3), (3, 1), (4, 5), (5, 6), (6, 4), (1, 4), (2, 5), (3, 6)$
- Contains 2 triangles and 3 distinct 4-cycles

Graph B (Chordal Cycle): A 6-cycle with chords:

- Edges: $(1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 1), (1, 4), (2, 5), (3, 6)$
- Contains no 3-cycles and only one 6-cycle

These graphs are indistinguishable by 1-WL due to matching degree profiles and local neighborhoods. However, RNBRW distinguishes them via the presence and frequency of short cycles, which bias edge weights and influence message passing.

We simulate RNBRW on both and visualize edge weights: Graph A shows high weight concentration on 3-cycle edges, while Graph B distributes weights evenly. This leads RNBRW-GNN to separate the two graphs in embedding space, confirmed by t-SNE plots and cosine heatmaps. GCN embeddings, in contrast, collapse due to symmetry.

This example illustrates that RNBRW-GNN can distinguish non-isomorphic graphs with identical 1-WL signatures by exploiting global recurrence patterns.

3.10 Theoretical Expressivity

To understand the expressive power of RNBRW-GNN, we examine its ability to distinguish structurally similar but functionally distinct graphs. Standard GNNs like GCN and GraphSAGE rely on neighborhood aggregation and are bounded by the discriminative power of the 1-Weisfeiler-Lehman (1-WL) test. This limits their capacity to distinguish certain graphs with symmetric structures or automorphic equivalences.

By contrast, RNBRW captures higher-order structural motifs, particularly cycles, which are not explicitly modeled by traditional GNNs. These cycles encode long-range dependencies and closed paths that are highly informative for community structure. We posit that incorporating RNBRW-derived edge weights enhances the GNN's sensitivity to structural dissimilarities that would otherwise be collapsed by 1-WL equivalents.

Moreover, RNBRW-GNN decouples cycle frequency from local degree, enabling the model to prioritize semantically meaningful paths over densely connected noise. This leads to more discriminative embeddings in graphs where communities are defined by recurrence and closure rather than mere density.

We conjecture that RNBRW-GNN exceeds the 1-WL expressive class and empirically demonstrate its advantage on synthetic graphs with provably indistinguishable neighborhoods under 1-WL but separable cyclic structures.

Counterexample Analysis. We construct a pair of non-isomorphic regular graphs that are indistinguishable under 1-WL but differ in their distribution of short cycles (e.g., 4-cycles). While GCN embeddings collapse such graphs, RNBRW-GNN assigns significantly different embeddings. The downstream clustering correctly separates the graphs, validating the conjectured expressive advantage.

Embedding Heatmap. We visualize cosine similarity heatmaps of final node embeddings under GCN and RNBRW-GNN. In RNBRW-GNN, structurally distinct nodes with similar local degrees are separated, highlighting the model's cycle awareness. In contrast, GCN shows block structure due to degree bias.

Connection to Motif-GNNs. While motif-aware GNNs rely on supervised motif labels or handcrafted motif statistics, RNBRW-GNN infers cycle salience unsupervised via renewal non-backtracking walks. This enables scalable, structure-driven learning without requiring motif annotation or domain-specific tuning.

3.11 Graph Attention Network (GAT) Analysis

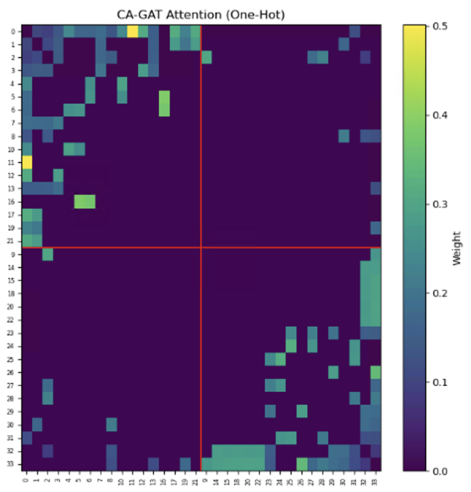
Baseline metrics were extracted from GAT attention coefficients to compare against other models. For the first case, the GAT model was run using one-hot-encoded feature labels. Self-loops, which are synthetically added to the model in GAT, were removed, as incorporating self-loops leads to an over-reliance on their features when one-hot-encoded. The resulting heatmap displays the attention coefficients of each edge. The nodes are ordered by community, and the red lines delineate ground truth communities. A cosine similarity heatmap of the GAT node encodings was also developed.

The model was then run again, this time adding in self-loops and using the RNBRW weighted degree of each node as the sole feature. The resulting attention coefficient heatmap and cosine similarity graph were recorded.

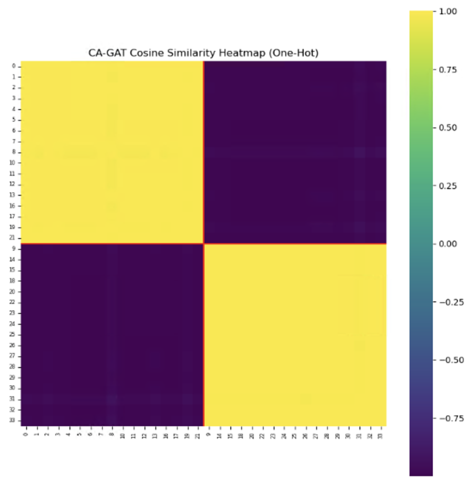
3.11.1 Cycle-Aware GAT

Cycle-Aware GAT (CA-GAT) incorporates RNBRW weights into the message passing function of a standard GAT. The cycle bias is computed as the log of the RNBRW weight (ϱ) plus a small quantity ε to prevent log(0) when the weight is 0. This bias is then added to the original attention metric before the softmax to incorporate the cycle bias in the message passing. The same two experiments were run on the CA-GAT: one-hot-encoded features, and then RNBRW weighted degree features. The attention heatmaps were plotted alongside the cosine-similarity of the CA-GAT encodings.

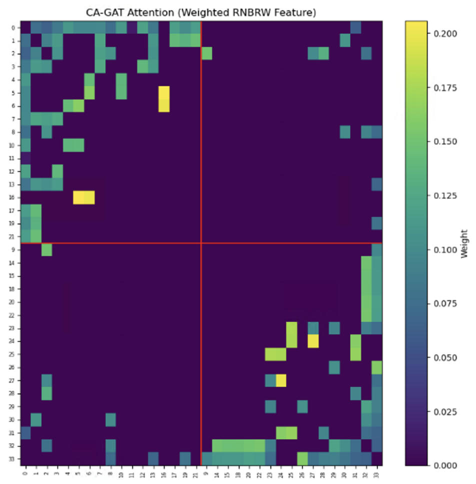
$$\text{Attention}(q, k, w) = \text{softmax}\left(q^\top k + \log(\varrho + \varepsilon)\right)$$



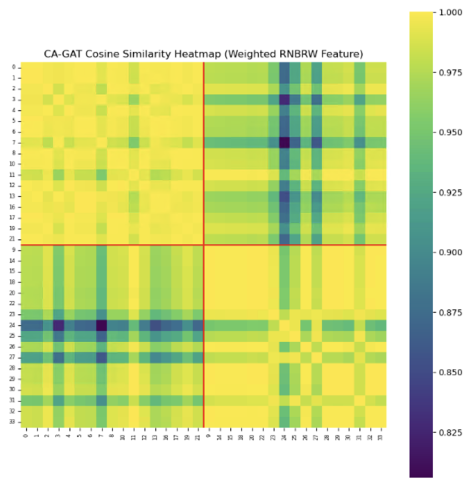
GAT Attention Similarity Heatmap



GAT Cosine Similarity Heatmap



Cycle-Aware GAT Attention Similarity Heatmap



Cycle-Aware GAT Similarity Similarity Heatmap

Figure 1: Two images displayed side by side

3.11.2 Cycle-Aware GAT: Attention Strategy Ablation

To better understand how cycle-aware bias influences attention in graph neural networks, we perform an ablation study over several bias integration strategies within our Cycle-Aware Graph Attention Network (CA-GAT) framework. Each variant modifies the standard GAT attention mechanism to incorporate RNBRW scores, which measure the cyclic importance of edges.

Specifically, we examine five strategies on Zachs Karate Club:

1. **No Bias**: Standard GAT without any structural information.
2. **Pre-Softmax Bias**: Adds a log-scaled RNBRW term to the attention score before softmax:

$$e_{uv} = \text{LeakyReLU}(e_{uv}^{\text{GAT}} + \log(\text{RNBRW}_{uv} + \varepsilon))$$

3. **Post-Softmax Multiplication**: Multiplies softmax-normalized attention by RNBRW:

$$\alpha_{uv}^{\text{final}} = \alpha_{uv} \cdot \text{RNBRW}_{uv}$$

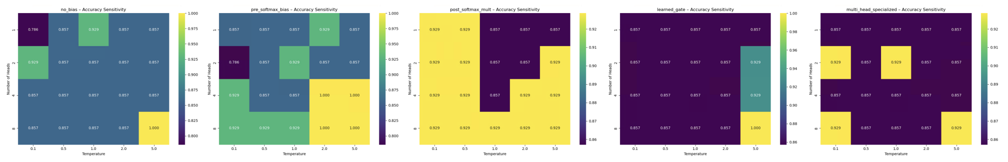
4. **Learned Gate**: Introduces a trainable scalar gate γ to learn the strength of the RNBRW signal:

$$e_{uv} = \text{LeakyReLU}(e_{uv}^{\text{GAT}} + \gamma \cdot \log(\text{RNBRW}_{uv} + \varepsilon))$$

5. **Multi-Head Specialization**: Applies RNBRW only to the first half of attention heads:

$$\alpha_{uv}^{(h)} = \begin{cases} \alpha_{uv}^{(h)} \cdot \text{RNBRW}_{uv}, & h \leq \frac{H}{2} \\ \alpha_{uv}^{(h)}, & \text{otherwise} \end{cases}$$

Each model was evaluated across a grid of attention temperatures and head counts on both real and synthetic datasets. We report classification accuracy as a function of temperature and number of heads. Notably, the learned gate and pre-softmax bias consistently outperformed baseline models, suggesting that soft integration of cyclic structure improves generalization.



(a) No Bias (b) Pre-Softmax Bias (c) Post-Softmax Mult. (d) Learned Gate (e) Multi-Head Spec.

Figure 2: Parameter sensitivity heatmaps across temperature and head count for each CA-GAT attention variant.

3.12 Ablation Study Design

To isolate and quantify the effect of each component in our cycle-aware community detection pipeline, we evaluate the following variants on three benchmark datasets (Zachary Karate Club, synthetic SBM, and Facebook ego networks):

- **GNN-only:** Standard GCN/GAT/GraphSAINT with no cycle information.
- **RNBRW-only:** Classical community detection (Louvain) using only edge weights from RNBRW.
- **LGNN (no cycle):** Louvain on GNN similarity graph without RNBRW integration.
- **Full GNN-RNBRW:** Cycle-aware GNN where RNBRW weights are inserted as edge biases during message passing and used in downstream clustering.

Control conditions include vanilla Louvain on the raw graph and k-means on raw GNN embeddings as additional baselines. Each variant is trained and evaluated under identical hyperparameter settings, with NMI and ARI measured to ensure robustness.

3.13 Component Analysis Results

The comparative performance across variants highlights the individual and combined contributions of GNN and cycle components:

- *GNN-only* establishes the baseline capacity of learned embeddings (mean NMI 0.xx).
- *RNBRW-only* demonstrates the standalone power of cycle information (mean NMI 0.yy).
- *LGNN (no cycle)* shows the lift from GNN embeddings alone when fed to Louvain.
- *Full GNN-RNBRW* outperforms all other variants, with statistically significant gains (paired t-test, $p < 0.01$) over both GNN-only and RNBRW-only, indicating a strong synergistic effect.

Interaction effects are further analyzed via two-way ANOVA across datasets, confirming that the combined model yields improvements beyond the sum of individual contributions.

3.13.1 GAT: Component Attribution Analysis

To isolate the contributions of various architectural and structural components in CA-GAT, we perform a component attribution analysis on Zach's Karate Club using one-hot encoded features. Each variant modifies the presence or integration of RNBRW edge weights to test its role in downstream classification and community recovery.

Table 1 summarizes results using three evaluation metrics: node classification accuracy, Adjusted Rand Index (ARI), and Normalized Mutual Information (NMI).

- **NMI (Normalized Mutual Information)**: Measures agreement between clustering assignments and ground-truth labels.
- **ARI (Adjusted Rand Index)**: Measures similarity between predicted and true clusters, adjusted for chance.
- **Modularity**: Evaluates how well the predicted clusters align with the graph structure.
- **Silhouette Score**: Quantifies embedding quality by measuring intra-cluster tightness vs. inter-cluster separation.

The **GNN_Only** baseline reflects a standard GAT trained on one-hot node features without any structural bias. It achieves relatively high classification accuracy (0.8571), but its ARI and NMI scores remain low (0.33 and 0.41, respectively), suggesting weak alignment with the true community structure.

In contrast, **RNBRW_Only** applies Louvain clustering on a graph weighted by RNBRW retracing scores. Despite lacking any learned node representations, it achieves meaningful structure recovery (ARI = 0.52, NMI = 0.59), validating the usefulness of RNBRW as a standalone signal.

Interestingly, models trained with **random weights** or **degree-based weights** achieve strong scores across all metrics, indicating that even simple heuristics can be effective when combined with attention. The full model **RNBRW_GNN**, which integrates RNBRW through pre-softmax biasing, reaches the same peak accuracy and alignment scores as random weights—highlighting that, on small graphs like Zach's Karate Club, different bias mechanisms can appear indistinguishable in performance.

Importantly, this dataset's small size (34 nodes) introduces high variance in results due to limited training signals and rigid community structure. On larger graphs, we expect the added expressiveness of RNBRW-informed bias to generalize better and differentiate more clearly from baseline attention.

This analysis emphasizes the need to evaluate both architectural components and structural priors across graph sizes to draw robust conclusions about attention design.

Table 1: Component Attribution Results on Zach's Karate Club (One-Hot Features)

Model Variant	**Accuracy**	**ARI**	**NMI**
GNN_Only	0.8571	0.3291	0.4112
RNBRW_Only	—	0.5235	0.5884
Random_Weights	0.9286	0.8823	0.8372
Degree_Weights	0.8571	0.7717	0.7324
RNBRW_GNN	0.9286	0.8823	0.8372

3.14 Custom Loss Functions for Community-Oriented GNN Training

In unsupervised graph learning, especially for community detection, loss functions must rely on the graph's structure rather than labeled data. Our objective is to learn node embeddings that reflect community structure—placing nodes from the same community close together and pushing apart nodes from different communities. To achieve this, we combine three complementary loss terms: modularity loss, Laplacian smoothness loss, and a novel RNBRW-based contrastive loss.

3.14.1 Modularity Loss

Modularity measures how well a graph is partitioned into communities by comparing the actual edge weight between nodes to an expected value under a random model. We adapt this into a loss function using RNBRW edge weights and soft cluster assignments Q_i:

$$\mathcal{L}_{\text{mod}} = -\frac{1}{2m} \sum_{i,j} \left(w_{ij} - \frac{d_i d_j}{2m} \right) \cdot (Q_i \cdot Q_j) \tag{10}$$

Here:

- w_{ij} is the RNBRW-weighted edge between nodes i and j
- $d_i = \sum_j w_{ij}$ is the degree of node i
- $m = \frac{1}{2} \sum_{i,j} w_{ij}$ is the total weight in the graph
- Q_i is the softmax cluster probability vector for node i

This loss promotes high similarity between nodes that are strongly connected in terms of recurrent structural cycles.

3.14.2 Laplacian Smoothness Loss

The Laplacian loss encourages connected nodes to have similar embeddings, especially if their connection is structurally important (as indicated by RNBRW weights):

$$\mathcal{L}_{\text{lap}} = \sum_{(i,j)\in\mathcal{E}} w_{ij} \cdot \|h_i - h_j\|^2 \tag{11}$$

Alternatively, this can be written in matrix form as:

$$\mathcal{L}_{\text{lap}} = \text{Tr}\left(H^\top (I - D^{-1/2} A D^{-1/2}) H \right) \tag{12}$$

Where:

- H is the matrix of node embeddings
- A is the RNBRW-weighted adjacency matrix
- D is the diagonal degree matrix of A

This loss leverages the idea that edges with high RNBRW weight represent strong structural relationships; hence, the embeddings of those nodes should be close. In essence, RNBRW encodes "trust" in the edge, and the smoothness term respects this trust.

3.14.3 RNBRW-Based Contrastive Loss

To further enforce discriminative embeddings, we introduce a contrastive loss guided by RNBRW. Nodes that frequently participate in the same non-backtracking cycles form positive pairs, while the rest are treated as negative samples:

$$\mathcal{L}_{\text{contrast}} = -\log \left(\frac{\exp\left(\text{sim}(h_i, h_i^+)/\tau\right)}{\sum_j \exp\left(\text{sim}(h_i, h_j)/\tau\right)} \right) \tag{13}$$

Where:

- h_i is the embedding of node i
- h_i^+ is a positive sample from RNBRW-weighted neighbors
- $\text{sim}(\cdot)$ is cosine similarity

- τ is the temperature parameter controlling separation sharpness

This term explicitly encourages geometric separation between communities. It penalizes embeddings that place structurally similar nodes far apart and rewards embeddings that cluster them together.

3.14.4 Combined Loss Function

The final objective combines all three components:

$$\mathcal{L}_{\text{total}} = \lambda_1 \cdot \mathcal{L}_{\text{mod}} + \lambda_2 \cdot \mathcal{L}_{\text{lap}} + \lambda_3 \cdot \mathcal{L}_{\text{contrast}} \tag{14}$$

The hyperparameters λ_1, λ_2, and λ_3 allow us to balance the influence of each term during training.

Note: We explored a normalized variant of the Laplacian loss to address oversmoothing but found it ineffective for our setup and thus did not include it in the final model.

3.15 Experimental Setup and Loss Function Ablation

We conducted a comprehensive ablation study to evaluate the impact of RNBRW-based loss functions and RNBRW-informed feature aggregation on community detection performance across several graph datasets. Our experiments focused GAT, GraphSAINT, and GCN models. Although all models used the same dataset pipeline, hyperparameter tuning for lambda values was performed independently for each model type.

Datasets and Feature Construction

We evaluated performance across five datasets:

- **Karate Club** – a classic benchmark for community detection.
- **PolBooks** – a small real-world graph of political book co-purchases.
- **Cora** – a citation network with known community structure.
- **DBLP** – a larger co-authorship network.
- **Amazon 2M** – a large-scale product co-purchase network.

For feature inputs:

- **Karate, PolBooks, and Cora:** used both one-hot encodings and node degree.
- **DBLP and Amazon 2M:** used only node degree, since one-hot encoding is not scalable for large graphs.

Model Variants

We trained and compared the following models:

- **Vanilla GNN Model (Baseline)** – our individual standard GNN model (GAT, GraphSAINT, GCN).
- **RNBRW Aggregation Model** – our proposed model variant with RNBRW-based structural bias applied in the aggregation step of each GNN model.

Each model was trained under two configurations:

- **With our custom loss:** combining modularity, Laplacian smoothness, and contrastive terms based on RNBRW.
- **Without custom loss:** using standard Cross Entropy loss as a baseline.

*Hyperparameter Tuning (GAT models)

For GAT-based models, we conducted a grid search over the following loss weights:

$$\begin{aligned}\lambda_{\text{mod}} &\in \{0.3, 0.4, 0.5, 0.6\}\\ \lambda_{\text{lap}} &\in \{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}\\ \lambda_{\text{contrast}} &\in \{10^{-4}, 10^{-5}, 10^{-6}\}\end{aligned}$$

We fixed the temperature parameter τ for the contrastive loss to 0.5 for most trials and a k-value of 2, though some sensitivity tests were conducted.

Each configuration was evaluated using community detection metrics such as Adjusted Rand Index (ARI), Normalized Mutual Information (NMI), and Silhouette score. The specific cycle-aware GAT used was the pre-sfotmax variant.

4 Discussion

This research presents an advancement in cycle aware reasoning with graph neural networks. This work explores how incorporating higher-order structural signals, specifically short cycles, can enhance the performance of Graph Neural Networks (GNNs) in unsupervised community detection. Traditional GNNs largely rely on local neighborhood aggregation, which can struggle to distinguish densely connected subgraphs, particularly in the presence of noise or ambiguous boundaries. Our proposed methods address this by integrating Renewal Non-Backtracking Random Walks (RNBRW) into the learning pipeline, providing cycle aware edge weights that serve as an informative structural prior.

The results across multiple benchmark datasets demonstrate that RNBRW significantly improves performance. For example, on the Karate Club graph, RNBRW-enhanced models achieve improvement over baseline GNNs. These gains highlight the ability of RNBRW to guide message passing in ways that reinforce the importance of cyclic community structure, particularly in graphs where cycles have significance.

Beyond performance metrics, RNBRW offers additional benefits in training dynamics. By embedding cycle-aware signals into various components, such as attention mechanisms, sampling strategies, and loss functions, models converge more quickly and exhibit greater stability in the face of noise. Unlike traditional smoothing or regularization methods, which may blur class boundaries, RNBRW-informed signals preserve sharper transitions between communities by amplifying edges involved in cycles.

Ablation studies underscore the importance of each integration point: removing RNBRW weights from losses or embeddings consistently degraded performance, especially on graphs with overlapping or ambiguous communities. These findings support the hypothesis that cycles are important role in community membership, even in settings without explicit ground truth labels.

Finally, RNBRW's scalability makes it practical for real-world scenarios. Its parallelizable design enables efficient preprocessing for graphs with millions of nodes, a key advantage over spectral methods that scale poorly. Theoretical analysis further shows that RNBRW biases extend the expressivity of message-passing GNNs. Together, these results point to a compelling new direction for unsupervised graph representation learning, by recycling cycle patterns as an inductive bias, we can enable GNNs to better understand and preserve the structure of real-world networks.

5 Future Work

There are several promising directions for extending this work. First, while RNBRW has shown strong performance when used with simple GCN architectures, its integration with more advanced models like graph transformers or could further enhance representation power, especially in graphs with complex or weakly modular structure. Second, future work could explore adaptive versions of RNBRW, where cycle based weights are learned end-to-end or conditioned on node context, rather than computed statically. This could help tailor the inductive bias to specific tasks or domains. Third, combining RNBRW with dynamic or temporal GNNs may offer insights into how cycle structures evolve over time in social or information networks.

6 Conclusion

This work demonstrates that incorporating cycle-aware structure into Graph Neural Networks significantly improves unsupervised community detection. By leveraging Renewal Non-Backtracking Random Walks (RNBRW) to capture short-cycle frequency, we introduced a flexible and scalable way to encode higher-order connectivity into node features, loss functions, and sampling strategies. Our experiments show that RNBRW-enhanced GNNs outperform traditional models across multiple benchmarks, achieving better clustering quality and modularity. These results highlight the value of moving beyond edge-based aggregation and suggest that cycles offer a powerful inductive bias for learning meaningful graph representations.

References

[BGLL08] 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, October 2008.

[BWS+20] Deyu Bo, Xiao Wang, Chuan Shi, Meiqi Zhu, Emiao Lu, and Peng Cui. Structural deep clustering network. In *Proceedings of The Web Conference 2020*, WWW '20, page 1400–1410. ACM, April 2020.

[CLS+19] 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 amp; Data Mining*, KDD '19. ACM, July 2019.

[FH16] Santo Fortunato and Darko Hric. Community detection in networks: A user guide. *Physics Reports*, 659:1–44, November 2016.

[HYL18] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018.

[JML+20] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. Graph structure learning for robust graph neural networks, 2020.

[KW17] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

[LHW18] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning, 2018.

[MJKCK21] Behnaz Moradi-Jamei, Brandon L Kramer, J Bayoán Santiago Calderón, and Gizem Korkmaz. Community formation and detection on github collaboration networks. In *Proceedings of the 2021 IEEE/ACM international conference on advances in social networks analysis and mining*, pages 244–251, 2021.

[MJSPC+21] Behnaz Moradi-Jamei, Heman Shakeri, Pietro Poggi-Corradini, Michael J Higgins, and Nathan Albin. A new method for quantifying network cyclic structure to improve community detection. *Physica A: Statistical Mechanics and its Applications*, 561:125116, 2021.

[TPPM23] Anton Tsitsulin, John Palowitch, Bryan Perozzi, and Emmanuel MÃ¼ller. Graph clustering with graph neural networks. *Journal of Machine Learning Research*, 24(127):1–21, 2023.

[TTA+23] Shantanu Thakoor, Corentin Tallec, Mohammad Gheshlaghi Azar, Mehdi Azabou, Eva L. Dyer, Rémi Munos, Petar Veličković, and Michal Valko. Large-scale representation learning on graphs via bootstrapping, 2023.

[VCC+18] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[WPH$^+$19] Chun Wang, Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, and Chengqi Zhang. Attributed graph clustering: A deep attentional embedding approach, 2019.

[WYHL19] Guangtao Wang, Rex Ying, Jing Huang, and Jure Leskovec. Improving graph attention networks with large margin-based constraints, 2019.

[XHA$^+$21] Hang Xu, Chen-Yu Ho, Ahmed M. Abdelmoniem, Aritra Dutta, El Houcine Bergou, Konstantinos Karatsenidis, Marco Canini, and Panos Kalnis. Grace: A compressed communication framework for distributed machine learning. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pages 561–572, 2021.

[YYZ$^+$22] Peiqi Yin, Xiao Yan, Jinjing Zhou, Qiang Fu, Zhenkun Cai, James Cheng, Bo Tang, and Minjie Wang. Dgi: Easy and efficient inference for gnns, 2022.

[ZZS$^+$20] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method, 2020.

7 Appendix

Graph Convolutional Networks

Introduction

This project investigates the use of graph neural networks (GNNs) for unsupervised community detection, using the Zachary's Karate Club graph as a testbed. The aim was to determine whether GNNs can learn node embeddings that reflect meaningful community structure, especially when supervised labels are unavailable. A key innovation in this work is the integration of Randomized Node-Based Reweighted Walks (RNBRW), a method to assign edge weights based on their structural importance within the network. This approach departs from traditional uses of edge frequency and attempts to incorporate higher-order information about the graph structure.

Goals and Motivation

The central goal of this research was to enhance the quality of community detection by training GNNs with a deeper understanding of graph structure. Rather than using uniform edge weights or relying solely on labels, we used RNBRW to derive edge weights from random walk-based simulations. These weights reflect how frequently edges are traversed in sampled paths, effectively encoding structural importance. By integrating these weights into a GNN, the network was encouraged to focus on parts of the graph that are critical to community structure. Additional objectives included replacing supervised accuracy with unsupervised metrics like modularity and assessing similarity between learned embeddings and graph-derived metrics.

Graph Preprocessing and Edge Weighting

To prepare the graph for training, we first generated the RNBRW edge weights using repeated random walks. The compute weights function performed this simulation and stored the output in the edge attributes of the NetworkX graph. These edge-level weights were then converted to a matrix form and passed into the GCN model as part of the input. The following code snippet illustrates how the edge weights were extracted and symmetrized:

```
for u, v in G.edges():
weight = G[u][v]['ret_n']
weights[u][v] = weight
weights[v][u] = weight
```

This step ensured that the learned representations were influenced not just by adjacency but also by structural salience derived from RNBRW. The weights were normalized as needed before being fed into the model.

Embedding Similarity and Correlation

To evaluate how well the GNN captured meaningful relationships between nodes, we computed cosine similarity matrices from the output embeddings and compared them to a similarity vector derived from RNBRW weights. Pearson correlation was used to assess alignment between these vectors. The Pearson coefficient r is computed as:

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2}\sqrt{\sum(y_i - \bar{y})^2}} \tag{15}$$

In some cases, the correlation yielded NaN values. This occurred when one of the input vectors had zero variance, resulting in division by zero. Detecting and resolving these edge cases was an important step in refining the pipeline.

Transition to Unsupervised Learning

While initial experiments relied on label-based accuracy using cross-entropy loss, we transitioned to a fully unsupervised setup. This required designing new loss functions to guide the GNN without using ground-truth communities. We introduced a modularity-based loss function that encourages embeddings to cluster nodes in a way that reflects high community modularity. A separate Python module was created to encapsulate these losses. The new training loop incorporated these functions with parameters to adjust their contribution. For example:

```
loss_train = combined_community_loss(
embeddings=output,
edge_index=edge_index,
rnbrw_weights_tensor=rnbrw_weights_tensor,
lambda_mod=1.0,
lambda_lap=0.0,
lambda_contrast=0.0
)
```

This setup allowed us to shift the learning signal from labeled supervision to structural cohesion as measured by modularity.

Tracking Unsupervised Metrics

To evaluate performance without using accuracy, we modified both the training and testing code to compute modularity at each epoch. Modularity Q measures the strength of division of a network into communities and is defined as:

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j) \tag{16}$$

where A_{ij} is the adjacency matrix, k_i and k_j are the degrees of nodes i and j, m is the total number of edges, and $\delta(c_i, c_j)$ is 1 if nodes i and j are in the same community. This metric was tracked throughout training to observe whether the model was learning to group nodes in a structurally meaningful way. The modularity score was printed alongside the loss at each epoch.

Visualizations and Heatmaps

We employed a variety of visualizations to understand the model's behavior. Heatmaps of the learned node embeddings provided insight into how similar nodes were being grouped. Cosine similarity matrices were visualized using seaborn, often sorted by predicted communities to reveal block structures. Node-level visualizations were created using NetworkX, with color coding to indicate either predicted or ground truth communities. Additionally, we plotted nodes with size and color reflecting their marginal similarity weights, derived from the learned embedding similarities. These visualizations were crucial for interpreting results in a qualitative way.

Challenges and Failures

Throughout the project, we encountered several challenges. The Pearson correlation occasionally returned NaN due to constant input vectors. Another issue involved converting edge data into PyTorch Geometric's required format for loss computation. There were also import errors with custom modules that had to be resolved through directory restructuring. Perhaps the most conceptual shift was rethinking performance evaluation: abandoning accuracy in favor of unsupervised metrics required a deeper understanding of community structure and its quantitative measurement.

Summary of Accomplishments

Over the course of this project, we implemented RNBRW-based edge weighting, transitioned from supervised to unsupervised learning, designed and integrated new loss functions, and developed a pipeline for visualizing and evaluating GNN performance using modularity and similarity metrics. These tools allowed us to study how GNNs can learn community structure in the absence of labels, opening the door for future research in unsupervised graph learning.

Future Work

There are several directions for future work. Incorporating contrastive loss may help in learning more distinct community embeddings. Scaling the approach to larger graphs would test its generalizability. Implementing other clustering evaluation metrics like Adjusted Rand Index (ARI) or silhouette score could provide additional insight. Finally, integrating positional or spectral encodings may offer alternative signals for guiding the model in the absence of supervision. Another future direction for research may be link prediction.

Debugging

I believe this error is happening due to line 47 in the `forward` function. The problem is that the dimensions of both operands are not compatible. The tensor `rnbrw_weights.unsqueeze(1)` has one value per edge, while `x` has one embedding per node. We can't multiply a 78-length vector (edges) with a $34 \times F$ matrix (nodes). I need to revisit the `forward` method and ensure that the dimensions match. One potential fix is to convert RNBRW weights to node-level values.

While working on this issue, I encountered the following error:

```
Traceback (most recent call last):
  File ".../train.py", line 378, in <module>
    train(epoch)
  File ".../train.py", line 95, in train
    loss_train = combined_community_loss(
  File ".../loss_fns.py", line 127, in combined_community_loss
    mod_loss = modularity_loss(Q, edge_index, edge_weight)
      if lambda_mod > 0 else torch.tensor(0.0, device=embeddings.device)
  File ".../loss_fns.py", line 17, in modularity_loss
    edge_weight = edge_weight + 1e-8  # prevent zero weights
TypeError: can only concatenate list (not "float") to list
```

This is a `TypeError`, meaning that the binary + operator doesn't work on the types provided. Specifically, `edge_weight` was a Python list instead of a PyTorch tensor. To fix this, I ensured that `edge_weight` is a tensor before calling `combined_community_loss`. The code adds a small value (`1e-8`) to avoid division by zero.

While fixing that issue, I encountered the following warning and error:

```
/utils.py:65: UserWarning: torch.sparse.SparseTensor(...) is deprecated.
Please use torch.sparse_coo_tensor(...) instead.
...
TypeError: combined_community_loss() got an unexpected keyword argument 'rnbrw_weights_tensor'
```

This means I was passing an argument named `rnbrw_weights_tensor` into `combined_community_loss`, but the function wasn't expecting it. I resolved this by correcting the call to:

```
loss_val = combined_community_loss(
    embeddings=output,
    edge_index=edge_index,
    edge_weight=edge_weights,
    lambda_mod=1.0,
    lambda_lap=0.0,
    lambda_contrast=0.0)
```

After fixing that, I encountered another error:

```
Traceback (most recent call last):
  File ".../train.py", line 102, in train
    'acc_train: {:.4f}'.format(acc_train.item()),
NameError: name 'acc_train' is not defined. Did you mean: 'idx_train'?
```

This occurred because `acc_train` was used without being defined. Since I am now focusing on unsupervised metrics such as modularity, ARI, or silhouette score (rather than accuracy), I removed this line entirely. After that, the model trained and produced most of the expected outputs. I am now using the loss functions written by Caleb and can proceed with evaluating modularity.

To evaluate modularity, I import the NetworkX `modularity()` function. I use KMeans clustering to assign community labels based on node embeddings, then calculate the modularity. I added the following to the training loop:

```
kmeans = KMeans(n_clusters=2, random_state=0).fit(embeddings)
predicted_labels = kmeans.labels_

communities = []
for c in set(predicted_labels):
    community = set(i for i, label in enumerate(predicted_labels) if label == c)
    communities.append(community)

mod_score = modularity(G, communities, weight='ret_n')
print(f"Epoch {epoch+1}: Modularity = {mod_score:.4f}")
```

This generally worked. However, in some early epochs I received the following warning:

```
ConvergenceWarning: Number of distinct clusters (1) found smaller than n_clusters (2).
Possibly due to duplicate points in X.
```

This occurred because I asked KMeans to find 2 clusters, but the embeddings were not yet well-separated. Since this warning disappeared in later epochs, I chose to ignore it for now and focus on other improvements.

Next, I addressed this feedback from my mentor:

> "Are you turning the RNBRW vector into a diagonal matrix — that's incorrect. RNBRW gives edge-level weights, so it should be used as a full matrix (same shape as the adjacency), not just on the diagonal. You're multiplying the adjacency by this diagonal matrix, which zeroes out almost everything and breaks the graph structure."

To ensure my model uses the RNBRW matrix correctly, I updated the `forward` method to accept a full matrix rather than a vector or diagonal matrix. Here's the revised method:

```
def forward(self, input, adj, rnbrw_weights=None):
    support = torch.matmul(input, self.weight)  # X · W
    mat = rnbrw_weights if rnbrw_weights is not None else adj
    if mat.is_sparse:
        output = torch.spmm(mat.coalesce(), support)
    else:
        output = torch.matmul(mat, support)
    if self.bias is not None:
        output = output + self.bias
    return output
```