Feature-Centric Unsupervised Node Representation Learning Without Homophily Assumption

Sunwoo Kim[1], **Soo Yong Lee**[1], **Kyungho Kim**[1], **Hyunjin Hwang**[1], **Jaemin Yoo**[2], **Kijung Shin**[1,2]

[1]Kim Jaechul Graduate School of AI, KAIST, Seoul, South Korea
[2]School of Electrical Engineering, KAIST, Daejeon, South Korea
{kswoo97, syleetolow, kkyungho, hyunjinhwang, jaemin, kijungs}@kaist.ac.kr

Abstract

Unsupervised node representation learning aims to obtain meaningful node embeddings without relying on node labels. To achieve this, graph convolution, which aggregates information from neighboring nodes, is commonly employed to encode node features and graph topology. However, excessive reliance on graph convolution can be suboptimal—especially in non-homophilic graphs—since it may yield unduly similar embeddings for nodes that differ in their features or topological properties. As a result, adjusting the degree of graph convolution usage has been actively explored in supervised learning settings, whereas such approaches remain underexplored in unsupervised scenarios. To tackle this, we propose FUEL, which adaptively learns the adequate degree of graph convolution usage by aiming to enhance intra-class similarity and inter-class separability in the embedding space. Since classes are unknown, FUEL leverages node features to identify node clusters and treats these clusters as proxies for classes. Through extensive experiments using 15 baseline methods and 14 benchmark datasets, we demonstrate the effectiveness of FUEL in downstream tasks, achieving state-of-the-art performance across graphs with diverse levels of homophily.

1 Introduction

Node embeddings, vector representations that capture the corresponding node information, enable the use of various machine learning models in tackling downstream tasks such as node classification and clustering. Among various approaches, unsupervised node representation learning—designed to obtain useful embeddings without label supervision—has gained significant attention (Grover and Leskovec 2016; Hou et al. 2022; Chen, Lei, and Wei 2024; Ju et al. 2023). This approach is cost-effective, as it eliminates the need for labeling efforts and often outperforms supervised methods in label-scarce scenarios (Veličković et al. 2019).

To capture input node features and topology, *graph convolution* (Kipf and Welling 2017; Wu et al. 2019; Lee et al. 2024), which aggregates and propagates features from a node's local neighborhood, is widely leveraged (He et al. 2023; Wang et al. 2024). Especially, it has demonstrated its effectiveness in homophilic graphs, where similar nodes are likely to be connected (Guo et al. 2023).

However, *excessively relying* on graph convolution can be suboptimal, particularly in non-homophilic graphs where dissimilar nodes are frequently connected. This limitation arises since graph convolution inherently promotes similarity in the embeddings of adjacent nodes, leading to highly similar embeddings for dissimilar nodes that are frequently linked.

While adjusting the degree of graph convolution usage has been widely studied in supervised settings, it remains largely underexplored in unsupervised scenarios. In supervised settings, specialized GNN architectures learn an adequate degree of graph convolution based on the downstream supervision. However, such supervision is absent in unsupervised learning settings, making it challenging to adjust the adequate degree of graph convolution usage.

To address this, we propose **FUEL** (**F**eature-centric **U**nsupervised node r**E**presentation **L**earning), a novel unsupervised node representation learning method designed without homophily assumption. In a nutshell, FUEL adaptively determines the degree of graph convolution usage to increase intra-class similarity and inter-class separability in the embedding space. Since explicit class information is unavailable, FUEL relies instead on node features, which offer class-relevant information independent of the graph topology. Specifically, nodes with similar features are expected to belong to the same class, naturally forming a distinct cluster in the feature space. Building on this basis and a specialized clustering scheme, FUEL learns the degree of graph convolution that coheres each cluster while separating different clusters—a strategy backed by both theoretical and empirical evidence. Subsequently, FUEL refines node embeddings from the selected level of graph convolution to further enhance separability among clusters.

Through extensive experiments including 15 baseline methods and 14 real-world benchmark graph datasets, we demonstrate the effectiveness of FUEL in two widely studied node-level downstream tasks: node classification and clustering. Notably, the superiority of FUEL holds across graphs with various graph-level homophily, achieving the best node classification and clustering performance on both the least homophilic (Texas) and the most homophilic (Photo) graphs. Our key contributions are summarized as follows:

- We introduce a clustering-based proxy for class separability applicable in unsupervised settings and demonstrate its effectiveness both empirically and theoretically (Section 3).

- We present FUEL, a novel unsupervised node representation learning method that adaptively learns the adequate degree of graph convolution through clustering (Section 4).
- We validate the effectiveness of FUEL through extensive experiments, demonstrating its superiority over baseline methods in 10 out of 14 settings (Section 5).

For reproducibility, code and datasets are available at https://github.com/kswoo97/unsupervised-non-homophilic.

2 Related Work and Preliminary

In this section, we provide the preliminaries of our research and review the relevant literature.

2.1 Preliminary

A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ is defined by a set of nodes $\mathcal{V} = \{v_1, v_2, \cdots, v_{|\mathcal{V}|}\}$, a set of edges $\mathcal{E} \subseteq \binom{\mathcal{V}}{2}$, and a node feature matrix $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$. Each edge $e_j \in \mathcal{E}$ is a node pair, and edges can also be represented by an adjacency matrix $\mathbf{A} \in \{0,1\}^{|\mathcal{V}| \times |\mathcal{V}|}$, where $\mathbf{A}_{i,j} = 1$ if and only if $\{v_i, v_j\} \in \mathcal{E}$. Each node $v_i \in \mathcal{V}$ is equipped with a feature vector $\mathbf{x}_i \in \mathbb{R}^d$, which together form the node feature matrix $\mathbf{X}$, where the $i-$th row of $\mathbf{X}$ is $\mathbf{x}_i$. Therefore, a graph can alternatively be written as $\mathcal{G} = (\mathbf{X}, \mathbf{A})$.

2.2 Related Work

Unsupervised node representation learning. Early unsupervised node representation learning methods primarily focus on encoding the topological structure of a given graph (Perozzi, Al-Rfou, and Skiena 2014; Grover and Leskovec 2016). On the other hand, recent approaches focus on the integration of node features and graph topology through self-supervised learning (Veličković et al. 2019; Thakoor et al. 2021). Typically, they use (1) contrastive approaches that contrast nodes across different views (Hassani and Khasahmadi 2020; You et al. 2020) or (2) generative approaches that reconstruct masked node features (Hou et al. 2022, 2023) and/or edges (Li et al. 2023; Kim et al. 2024). Recent work adapts these approaches to perform well even in non-homophilic graphs, where nodes with dissimilar features and/or structural properties are often connected. (Liu et al. 2023; Wang et al. 2024). Some of them leverage multiple graph filters and contrast the embeddings obtained through different filters (Chen, Lei, and Wei 2024). Another line of work focuses on filtering non-homophilic edges using feature and/or topological similarity, followed by the application of existing self-supervised learning techniques (Yang and Mirzasoleiman 2024).

Adjusting the degree of graph convolution usage. In (semi-)supervised graph learning, adjusting the degree of graph convolution has been widely explored, often through the design of specialized graph neural networks (GNNs) (Zheng et al. 2022; Yu et al. 2024). Notably, GPR-GNN (Chien et al. 2021) and GADC (Luan et al. 2022) achieve this by learning a distinct coefficient for each k-th power of the normalized adjacency matrix used in graph convolution. However, since these coefficients are tuned by minimizing downstream task losses (e.g., cross-entropy for node classification), learning the adequate degree of graph convolution usage remains largely unexplored in unsupervised settings.

Clustering-based learning. Clustering, a process of grouping data based on their characteristics, has effectively guided unsupervised, particularly self-supervised, representation learning in computer vision (Khorasgani, Chen, and Shkurti 2022; Walawalkar and Garrido 2024). These approaches typically cluster visually similar images and enable image encoders to capture these similarities through pseudo-label classification (Asano, Rupprecht, and Vedaldi 2020) and/or contrastive learning (Caron et al. 2020).

3 Proposed Proxy for Class Separability

Recall that the core idea of FUEL, our proposed method, is to learn an adequate degree of graph convolution usage that increases the embeddings' ability to distinguish classes, which we call *class separability*. In this section, we introduce its proxy: *latent-class separability*. We begin by presenting the high-level motivation and concept of latent-class separability (Section 3.1), and then demonstrate its effectiveness as a proxy for class separability (Section 3.2).

3.1 Motivation

Goal and challenge. As discussed above, we regard adequate graph convolution usage as one that improves class separability in the embedding space. This property makes the learned embeddings well-suited for various popular downstream tasks, including node classification and clustering. However, since we consider unsupervised settings, we are not given any node labels. Therefore, we need a proxy measure for class separability.

Proposed proxy. We tackle this challenge by focusing on node features, which inherently reflect node classes. Nodes of the same class tend to share similar features, while those from different classes exhibit distinct ones. Therefore, nodes naturally form clusters in the feature space, which we term *latent classes*. With an adequate degree of graph convolution, features within the same class likely become more similar, enhancing the separability of latent classes. Based on this, we claim that *latent-class separability*—the degree to which each latent class is internally cohesive and distinct from others—can serve as an effective proxy for class separability.

3.2 Effectiveness of Latent-Class Separability

In this subsection, we first empirically show, using real-world graph datasets, that latent-class separability can serve as an effective proxy for class separability. We then give a theoretical justification for using latent-class separability as a proxy for class separability by proving that, under some conditions, latent-class separability is equivalent to class separability.

Empirical analysis setup. We analyze the correlation between class separability and latent-class separability across different degrees of graph convolution usage. To this end, we control the degree through the following graph convolution function, which produces node embeddings $\mathbf{Z} \in \mathbb{R}^{|\mathcal{V}| \times d}$ as: $\mathbf{Z} = \alpha_0 \mathbf{X} + \alpha_1 \hat{\mathbf{A}} \mathbf{X} + \alpha_2 \hat{\mathbf{A}}^2 \mathbf{X}$, where $\hat{\mathbf{A}}$ and $\hat{\mathbf{A}}^2$ denote the row-wise normalized versions of $\mathbf{A}$ and $\mathbf{A}^2$, respectively; the coefficients satisfy $0 \leq \alpha_i \leq 1, \forall i \in \{0,1,2\}$, and $\sum_{i=0}^{2} \alpha_i = 1$. We obtain 100 embeddings with varying degrees of graph convolution by, for each embedding: (1)

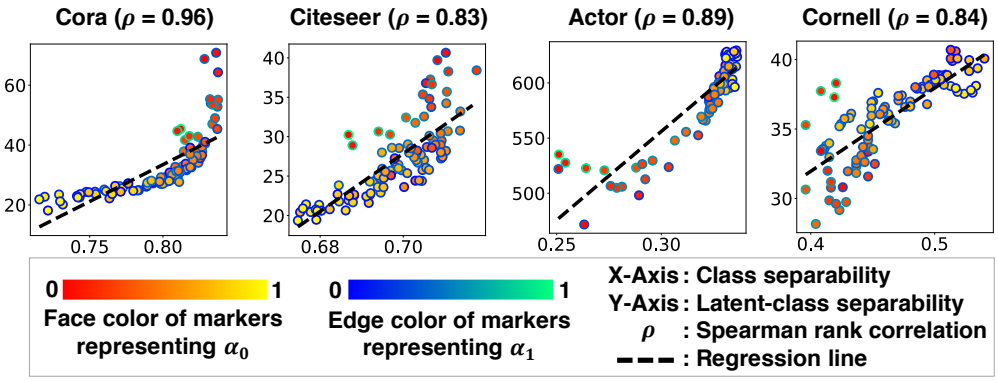


Figure 1: Effectiveness of latent-class separability as a proxy for class separability. Varying the graph convolution coefficients induces different degrees of graph convolution usage in the embeddings, which in turn leads to varying class separability. Notably, latent-class separability, our proposed proxy, strongly correlates with the actual class separability.

sampling three scalars uniformly at random from $[0, 1]$, (2) normalizing them by dividing each by their sum, and (3) assigning the resulting values to α_0, α_1, and α_2, respectively. In Section D.6 of the supplementary material, we further analyze representative graph convolution functions.[1]

We measure the class separability of an embedding by training a linear classifier on that embedding and reporting its accuracy on test nodes; additional metrics that directly use node labels are in Appendix D.11 of (Kim et al. 2026), yielding the same result. In contrast, we measure the latent-class separability by using the Calinski–Harabasz index (Caliński and Harabasz 1974), an unsupervised cluster quality metric that measures the ratio of mean inter-latent-class distance to mean intra-latent-class distance. Details on the Calinski–Harabasz index and its computation are provided in Section D.2 of the supplementary material. We use two homophilic graphs (Cora and Citeseer) and two non-homophilic graphs (Actor and Cornell), whose details are provided in Section B of the supplementary material. Nodes are split into 10% for training and 90% for testing.

Empirical analysis result. As shown in Figure 1, the latent-class separability of the embeddings is strongly and positively correlated with their class separability. Specifically, across both homophilic and non-homophilic datasets, (1) Spearman rank correlation values exceed 0.82, and (2) the fitted linear regression lines have positive slopes. These results suggest that latent-class separability is an effective proxy for class separability in real-world graph datasets.

Theoretical analysis setup. We theoretically show that under certain conditions, if one degree of graph convolution usage yields greater class separability than another, the same ordering holds for their latent-class separability. We consider two non-empty, disjoint node sets $\mathcal{C}_0 \cup \mathcal{C}_1 = \mathcal{V}$ of equal size (i.e., $|\mathcal{C}_0| = |\mathcal{C}_1|$), with each set corresponding to a distinct class. Node features $x_i, \forall v_i \in \mathcal{V}$ are generated from a Gaussian distribution, based on the class to which each node belongs.[2] Formally, the following holds: $x_i \sim \mathcal{N}(\mu, \sigma^2), \forall v_i \in \mathcal{C}_0$ and $x_j \sim \mathcal{N}(-\mu, \sigma^2)$ for $v_j \in \mathcal{C}_1$. Each node v_i has n neighbors, denoted by $N(v_i) \subset \mathcal{V}$, of which n_0 belong the same class of v_i and $n_1 = n - n_0$ belong to a different class. We define the embedding of v_i obtained via graph convolution as $z_i = \left((1-w)x_i + w\frac{1}{n}\left(\sum_{v_j \in N(v_i)} x_j\right)\right)$, where $w \in [0, 1]$ denotes the degree of graph convolution usage.

We denote the class separability of z as $\texttt{CS}(z; n, n_0, w)$, which is defined as one minus the Bayes error rate of the Bayes classifier (Bishop and Nasrabadi 2006) using the corresponding feature as an input (i.e., $\mathbb{E}_x[\mathbb{I}[P(v_i \in \mathcal{C}(v_i)|z_i) > P(v_i \notin \mathcal{C}(v_i)|z_i)]]$, where $\mathcal{C}(v_i)$ is the class of v_i and $P(v_i \in \mathcal{C}|x_i)$ represents the probability assigned by the Bayes classifier that v_i belongs to the class $\mathcal{C}(v_i)$, given its embedding z_i). In addition, we denote the latent-class separability of z as $\texttt{LCS}(z; n, n_0, w)$, which extends the Calinski-Harabasz index (Caliński and Harabasz 1974) as follows:

$$\frac{(\mathbb{E}_x[z_i|v_i \in \mathcal{C}_0] - \mathbb{E}_x[z_i|v_i \in \mathcal{C}_1])^2}{\frac{\mathbb{E}_x[(z_i - \mathbb{E}[z_i|v_i \in \mathcal{C}_0])^2|v_i \in \mathcal{C}_0] + \mathbb{E}_x[(z_i - \mathbb{E}[z_i|v_i \in \mathcal{C}_1])^2|v_i \in \mathcal{C}_1]}{2}}. \quad (1)$$

Theoretical analysis result. Our theoretical finding is summarized in the following theorem:

Theorem 1 (Effectiveness of latent-class separability)**.** *If* $n_0 = n$ *or* $w, w' \in [\max(\frac{n-2n_0}{2n-2n_0}, 0), 1]$ *hold, the following holds:* $\texttt{CS}(z; n, n_0, w) > \texttt{CS}(z; n, n_0, w')$ *if and only if* $\texttt{LCS}(z; n, n_0, w) > \texttt{LCS}(z; n, n_0, w')$.

Proof. Refer to Section A of (Kim et al. 2026). □

As shown in Theorem 1, the ordering of class separability between two degrees of graph convolution usage holds if and only if the same ordering holds for their latent-class separability, given that certain conditions for w and w' are met.

[1] We analyze APPNP (Gasteiger, Bojchevski, and Günnemann 2019) and GPR-GNN (Chien et al. 2021), and the results obtained using them are consistent with those presented in Section 3.2. Detailed results are in Section D.6 of the supplementary material.

[2] While our theoretical analysis focuses on two uniform-sized classes for an intuitive and rigorous theoretical analysis, we empirically demonstrate in Figure 1 and Appendix D.10 of (Kim et al. 2026) that latent-class separability remains an effective proxy for class separability in multi-class, imbalanced real-world graphs.

This theoretical result supports the effectiveness of latent-class separability as a proxy for class separability, indicating that the degree of graph convolution usage leading to better class separability can be effectively inferred from latent-class separability—especially in unsupervised settings.

4 Proposed Method

In this section, we introduce FUEL (**F**eature-centric **U**nsupervised node r**E**presentation **L**earning), a novel unsupervised node representation learning method. Building on the analyses provided in Section 3, FUEL learns the appropriate degree of graph convolution usage by aiming to enhance the latent-class separability. To this end, FUEL leverages clusters identified by a specialized clustering scheme (Section 4.1) as latent classes. Then, FUEL further enhances the latent-class separability of node embeddings from the selected degree of graph convolution usage through an additional refinement step (Section 4.2). Refer to Figure 2 for a pictorial overview of FUEL.

4.1 Step 1: A Specialized Clustering Scheme

FUEL employs a specialized clustering scheme to learn the adequate degree of graph convolution usage. In this scheme, an adaptive graph convolution model is employed, containing learnable parameters that directly control the impact of graph convolution on the resulting node embeddings. These learnable parameters are optimized to maximize the separability of clusters in the resulting embedding space.

Adaptive graph convolution model. As an adaptive graph convolution model, we use Eq. (2), where a learnable weight is assigned to each graph-convolution-based embedding term:

$$\mathbf{X}^* = (\alpha_0\mathbf{X} + \alpha_1\tilde{\mathbf{A}}\mathbf{X} + \alpha_2\tilde{\mathbf{A}}^2\mathbf{X}) \in \mathbb{R}^{|\mathcal{V}|\times d}, \quad (2)$$

where $\alpha_0, \alpha_1, \alpha_2 \in [0,1]$ and $\alpha_0 + \alpha_1 + \alpha_2 = 1$ hold, and $\mathbf{X}$ denotes the node feature matrix provided by the dataset. Note that Eq. (2) has the same functional form of the graph convolution function used in Section 3. We learn three scalars $c_0, c_1, c_2 \in \mathbb{R}$, which become $\alpha_0, \alpha_1, \alpha_2 \in \mathbb{R}$ via the softmax function (i.e., $\alpha_k = \frac{\exp(c_k)}{\sum_{j=0}^{2}\exp(c_j)}, \forall k \in \{0,1,2\}$).

Training for clustering. Our goal is to optimize the learnable weights in Eq. (2). To this end, inspired by entropy-based deep clustering methods in computer vision (Ghasedi Dizaji et al. 2017; Zhao and Mac Aodha 2023; Huang, Gong, and Zhu 2020), we incorporate entropy-based losses designed to achieve the following objectives: ensuring each node has a high cluster-assignment score to a single cluster ($\mathcal{L}_1$), and encouraging the expected cluster assignment distribution to be close to uniform over clusters to prevent nodes from collapsing into a single cluster ($\mathcal{L}_2$). Furthermore, based on our analyses in Section 3, we employ a distance loss to increase the mean inter-group distances over the mean intra-group distances, enhancing the separability of the group ($\mathcal{L}_3$). Specifically, each $c-$th cluster has a learnable centroid vector, denoted by $\mathbf{t}_c \in \mathbb{R}^d$, and the cluster-assignment score of node v_i to the $c-$th cluster, denoted by p_{ic}, is defined as: $p_{ic} = \frac{\exp((\mathbf{x}_i^*)^T\mathbf{t}_c)}{\sum_{k\in[C]}\exp((\mathbf{x}_k^*)^T\mathbf{t}_c)}$, where C is the number of clusters. Then, each loss is defined formally as follows:

$$\mathcal{L}_1 = -\frac{1}{|\mathcal{V}|}\sum_{v_i\in\mathcal{V}}\sum_{c\in[C]} p_{ic}\log p_{ic}, \quad (3)$$

$$\mathcal{L}_2 = \sum_{c\in[C]}\bar{p}_c\log\bar{p}_c, \text{ where } \bar{p}_c = \frac{1}{|\mathcal{V}|}\sum_{v_i\in\mathcal{V}} p_{ic}, \quad (4)$$

$$\mathcal{L}_3 = \exp\Big(\sum_{v_i,v_j\in\mathcal{V}^+}\frac{d(\mathbf{x}_i^*,\mathbf{x}_j^*)}{|\mathcal{V}^+|} - \sum_{v_k,v_\ell\in\mathcal{V}^-}\frac{d(\mathbf{x}_k^*,\mathbf{x}_\ell^*)}{|\mathcal{V}^-|}\Big), \quad (5)$$

where e is an exponential function, $d(\mathbf{x},\mathbf{y}) = \|\mathbf{x}-\mathbf{y}\|_2$, $\mathcal{V}^+ = \{\{v_i,v_j\} : \mathcal{T}(v_i) = \mathcal{T}(v_j), \{v_i,v_j\} \in \binom{\mathcal{V}}{2}\}$, $\mathcal{T}(v_i) = \arg\max_{c\in[C]} p_{ic}$, and $\mathcal{V}^- = \binom{V}{2} \setminus \mathcal{V}^+$. The final clustering loss is defined as $\mathcal{L}_{clus} = \mathcal{L}_1 + \mathcal{L}_2 + \lambda\mathcal{L}_3$, where λ is a loss coefficient for the distance loss term. All parameters (i.e., c_0, c_1, c_2 and $\mathbf{t}_c, \forall c \in [C]$) are optimized using gradient descent aiming to minimize $\mathcal{L}_{clus}$. After cluster training, using optimized weights, which are denoted as α_0^*, α_1^*, and α_2^*, we derive intermediate embeddings $\mathbf{H}$ as:

$$\mathbf{H} = (\alpha_0^*\mathbf{I} + \alpha_1^*\tilde{\mathbf{A}} + \alpha_2^*\tilde{\mathbf{A}}^2)\mathbf{X} \in \mathbb{R}^{|\mathcal{V}|\times d}. \quad (6)$$

4.2 Step 2: Improving Latent-Class Separability

Recall that our adaptive graph convolution model (Eq. (2)) is designed to learn an adequate degree of graph convolution usage. However, due to its simplicity (i.e., limited expressiveness), it may not be optimal for producing embeddings that maximize latent-class (i.e., cluster) separability. To enhance latent-class separability, we refine the intermediate node embeddings $\mathbf{H}$ (Eq. (6)) obtained from Step 1 (Section 4.1) using a refinement model.

Refinement model. As a refinement model that transforms the intermediate embeddings $\mathbf{H}$ into the final embeddings $\mathbf{Z}$, we use a feed-forward neural network f_θ. Moreover, a skip connection technique is incorporated to preserve the essential information encoded in $\mathbf{H}$. Consequently, the final node embeddings $\mathbf{Z}$ are defined as $\mathbf{Z} = f_\theta(\mathbf{H}) + \mathbf{H}$. During refinement, the intermediate embeddings $\mathbf{H}$ remain fixed, and only the parameters θ of the refinement model are optimized.

Training for refinement. Given intermediate node embeddings $\mathbf{H}$, we enhance their cohesion within each latent class (i.e., each cluster) while ensuring distinctiveness between different latent classes. Specifically, we enhance the similarity of embeddings for nearest neighbors in the intermediate embedding space (i.e., $\mathbf{H}$ space) since they are likely to belong to the same latent class due to the latent-class separability in the $\mathbf{H}$ space. Formally, we denote a set of N nearest neighbors of each node v_i in the $\mathbf{H}$ space as $\mathcal{V}_i'$. In addition, the set of nearest node pairs is denoted as $\mathcal{V}_+' = \{\{v_i,v_j\} : v_j \in \mathcal{V}_i', v_i \in \mathcal{V}\}$ and a set of negative pairs is denoted as $\mathcal{V}_-' = \binom{\mathcal{V}}{2} \setminus \mathcal{V}_+'$. We train f_θ to decrease the distance between node pairs in $\mathcal{V}_+'$ and increase the distance between node pairs in $\mathcal{V}_-'$.

As the objective function, we employ an exponential function, resulting in the overall loss function being a geometric average. We provide a detailed description of the advantage of using this loss function in Section D.3 of the supplementary

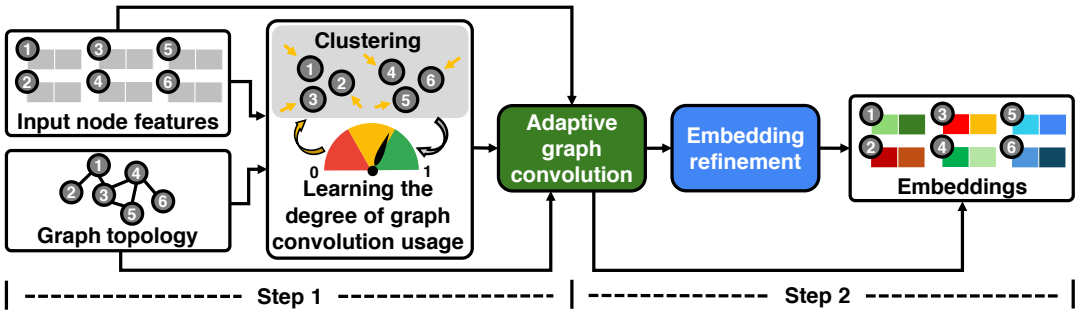


Figure 2: Overview of FUEL. FUEL consists of two steps. In Step 1, FUEL learns an adequate degree of graph convolution usage through the proposed clustering scheme and an adaptive graph convolution model. In Step 2, FUEL improves the cohesion of each latent class while preserving its distinction from others through embedding refinement.

material. Formally, the overall loss function for refinement is defined as follows:

$$\mathcal{L}_{dist} = \exp\Big(\sum_{v_i, v_j \in \mathcal{V}'_+} \frac{d(\mathbf{z}_i, \mathbf{z}_j)}{\tau |\mathcal{V}'_+|} - \sum_{v_k, v_\ell \in \mathcal{V}'_-} \frac{d(\mathbf{z}_k, \mathbf{z}_\ell)}{\tau |\mathcal{V}'_-|} \Big), \quad (7)$$

where $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2$ and τ is a temperature hyperparameter. The parameters of the refinement model are trained using gradient descent aiming to minimize $\mathcal{L}_{dist}$ (Eq. (7)). We provide a complexity analysis of FUEL in Section D.4 of the supplementary material.

5 Experiments

In this section, we evaluate the effectiveness of FUEL in several downstream tasks. To this end, we answer the following four research questions:

- **RQ1.** How effective are the node embeddings obtained by FUEL in the node classification task?
- **RQ2.** How effective are the node embeddings obtained by FUEL in the clustering task?
- **RQ3.** Does FUEL achieve its design goal?
- **RQ4.** Are all the key components of FUEL essential for achieving high performance?

5.1 Experimental Settings

Datasets. We evaluate FUEL on 14 real-world graph datasets, comprising 8 non-homophilic graphs (edge homophily (Zhu et al. 2020) < 0.5) and 6 homophilic graphs (edge homophily > 0.5). These datasets span seven diverse domains, with varying sizes (ranging from 183 to 167,343 nodes) and a wide range of homophily levels (edge homophily from 0.11 to 0.85).[3] Additional dataset details and results for other versions of certain datasets (Chameleon and Squirrel) are in Section B and D.9 of (Kim et al. 2026), respectively.

Baseline methods. For comparison, we use input node features alone (denoted as **Naive X**) and 14 unsupervised node representation learning baseline methods. These consist of 4 contrastive learning approaches (denoted as **CL**), 3 generative self-supervised learning approaches (denoted as **Gen**), and 7 non-homophilic graph unsupervised representation learning approaches (denoted as **Non-homophilic**). Each category consists of the following:

- **CL:** DGI (Veličković et al. 2019), GraphCL (You et al. 2020), MVGRL (Hassani and Khasahmadi 2020), and BGRL (Thakoor et al. 2021).
- **Gen:** GAE (Kipf and Welling 2016), GraphMAE (Hou et al. 2022), and MaskGAE (Li et al. 2023).
- **Non-homophilic:** HGRL (Chen et al. 2022), DSSL (Xiao et al. 2022), GREET (Liu et al. 2023), NeCo (He et al. 2023), PolyGCL (Chen, Lei, and Wei 2024), HeterGCL (Wang et al. 2024), and HLCL (Yang and Mirzasoleiman 2024).

Training and evaluation. To assess each method, we first obtain node embeddings using it, and then we use the embeddings as input features to perform downstream tasks, specifically node classification and clustering. Moreover, we provide qualitative analysis via embedding visualizations in Appendix D.14 of (Kim et al. 2026). For FUEL, we choose the number of clusters to equal the number of node classes in the dataset; Appendix D.12 of (Kim et al. 2026) further confirms that FUEL is not sensitive to this choice. For training/validation/test splits, we use fixed splits provided in PyG (Fey and Lenssen 2019). For datasets where fixed splits are not provided in PyG, we follow the node partitioning strategy proposed in the original works that introduced these datasets. Further details for the splits are in Section B.3 of the supplementary material. For each dataset–method pair, we run 10 trials, varying the model initialization and dataset splits. If a dataset provides only a single fixed split, we use the same split for all experiment trials. We perform the hyperparameter tuning with the validation set and use the configurations that give the best validation performance for the evaluation. Further details regarding the hyperparameter tuning are in Section C.2 of the supplementary material.

5.2 RQ1. Node Classification Results

In this section, we evaluate the effectiveness of each method in the node classification task.

Settings. We train an MLP classifier using node embedding obtained by each method as input features and cross-entropy

[3] For large-scale graphs with $|\mathcal{V}| > 4 \times 10^4$, we employ several scalable techniques, which are detailed in Section D.5 of the supplementary material.

	Datasets	Squirrel	Actor	Wis.	Cornell	Texas	Cham.	Penn94	Flickr	Cora	Citeseer	Pubmed	Photo	Comp.	Arxiv	A.R.
	Homophily	0.217	0.220	0.155	0.111	0.057	0.247	0.483	0.322	0.825	0.717	0.792	0.849	0.802	0.635	-
	Naive $\mathbf{X}$	34.0 (1.9)	35.4 (0.9)	79.0 (5.0)	71.1 (3.4)	75.1 (4.9)	49.8 (1.3)	73.7 (0.4)	49.9 (0.1)	58.5 (0.7)	59.0 (1.8)	72.8 (0.1)	88.5 (0.5)	81.8 (0.6)	58.9 (0.1)	14.0
CL	DGI	42.2 (1.2)	29.8 (1.5)	58.6 (6.0)	48.9 (6.8)	65.9 (4.6)	59.8 (2.5)	67.6 (0.8)	51.8 (0.3)	82.4 (0.6)	71.4 (1.4)	79.4 (0.8)	92.8 (0.6)	87.5 (0.5)	71.2 (0.3)	11.7
CL	GraphCL	49.1 (1.0)	30.7 (0.6)	60.6 (6.6)	42.4 (7.8)	65.0 (4.8)	64.5 (2.6)	O.O.M.	O.O.M.	82.4 (0.8)	71.3 (0.9)	83.4 (0.5)	93.0 (0.5)	90.3 (0.3)	O.O.M.	10.3
CL	MVGRL	49.1 (1.0)	30.0 (1.4)	69.7 (4.6)	47.8 (6.7)	69.7 (4.6)	57.1 (2.2)	O.O.M.	51.6 (0.3)	82.0 (1.2)	72.1 (0.8)	78.8 (0.9)	93.0 (0.4)	88.0 (0.5)	O.O.M.	12.1
CL	BGRL	55.0 (0.6)	30.9 (1.2)	59.4 (6.2)	49.2 (5.0)	64.3 (3.8)	66.8 (1.8)	74.2 (0.7)	53.1 (0.1)	82.1 (1.2)	71.2 (0.6)	81.0 (0.1)	93.2 (0.3)	90.0 (0.1)	71.8 (0.2)	8.4
Gen	GAE	47.8 (1.7)	30.3 (0.8)	54.3 (7.0)	43.5 (1.1)	67.0 (5.5)	57.9 (2.1)	O.O.M.	O.O.M.	81.9 (1.0)	69.1 (1.3)	76.6 (1.6)	89.0 (0.3)	93.0 (0.3)	O.O.M.	14.0
Gen	GraphMAE	45.8 (1.0)	29.8 (1.0)	59.6 (6.1)	50.8 (8.3)	65.9 (7.6)	66.2 (2.4)	59.1 (0.4)	51.7 (0.2)	83.5 (0.9)	73.0 (1.0)	80.8 (1.0)	92.6 (0.3)	88.8 (0.6)	71.3 (0.3)	9.7
Gen	MaskGAE	48.0 (1.0)	29.0 (1.4)	60.4 (0.0)	46.8 (7.4)	70.5 (5.6)	64.6 (1.4)	63.5 (0.5)	53.2 (0.2)	82.6 (0.9)	71.4 (0.7)	80.4 (1.8)	93.8 (0.3)	89.6 (0.8)	70.7 (0.1)	9.5
Non-homophilic	HGRL	46.3 (2.1)	37.3 (1.0)	82.2 (3.1)	74.4 (4.9)	83.4 (5.8)	61.1 (1.9)	O.O.M.	O.O.M.	80.2 (0.7)	70.7 (1.1)	78.8 (0.8)	92.8 (0.1)	88.1 (0.3)	O.O.M.	10.6
Non-homophilic	DSSL	53.5 (1.4)	29.2 (1.1)	60.0 (5.3)	45.1 (7.6)	64.1 (4.5)	68.5 (2.1)	O.O.M.	O.O.M.	82.0 (0.8)	66.0 (1.3)	77.3 (1.4)	92.8 (0.3)	89.7 (0.3)	O.O.M.	13.0
Non-homophilic	GREET	49.2 (2.0)	37.6 (1.3)	84.1 (4.3)	76.8 (4.4)	81.1 (5.7)	62.8 (1.3)	O.O.M.	O.O.M.	83.3 (0.5)	72.5 (0.8)	79.2 (0.7)	92.8 (0.3)	88.3 (0.3)	O.O.M.	7.8
Non-homophilic	NeCo	45.3 (1.3)	30.5 (1.1)	56.4 (5.4)	54.3 (6.2)	63.5 (5.0)	58.3 (2.3)	O.O.M.	O.O.M.	82.3 (0.8)	68.5 (1.2)	80.7 (0.7)	92.4 (0.2)	89.1 (0.3)	O.O.M.	13.1
Non-homophilic	HLCL	41.3 (1.7)	28.7 (1.2)	60.2 (4.4)	44.5 (7.2)	58.4 (3.7)	53.1 (2.4)	O.O.M.	O.O.M.	81.1 (1.2)	70.3 (0.7)	80.2 (1.3)	91.5 (0.3)	87.3 (0.3)	O.O.M.	15.5
Non-homophilic	PolyGCL	56.7 (1.5)	35.0 (1.0)	83.3 (3.1)	74.8 (6.2)	80.5 (6.3)	70.7 (2.1)	O.O.M.	O.O.M.	82.6 (1.2)	71.9 (0.7)	78.8 (1.0)	91.3 (0.4)	87.4 (0.6)	O.O.M.	9.1
Non-homophilic	HeterGCL	42.1 (1.1)	36.5 (1.4)	82.4 (3.6)	71.6 (4.1)	78.9 (5.6)	58.9 (1.7)	O.O.M.	O.O.M.	82.5 (1.0)	71.9 (1.1)	81.5 (0.6)	93.0 (0.3)	87.6 (0.6)	O.O.M.	9.2
Variants	w/o Step 1	46.6 (1.4)	34.2 (0.8)	64.3 (7.3)	54.9 (6.4)	63.2 (5.6)	52.1 (2.0)	74.3 (0.5)	50.8 (0.1)	83.5 (0.2)	73.3 (0.6)	79.4 (0.5)	93.9 (0.4)	90.1 (0.3)	71.7 (0.2)	8.3
Variants	w/o Step 2	56.6 (1.7)	36.2 (0.9)	79.0 (5.2)	70.8 (3.8)	75.1 (5.0)	70.7 (1.6)	75.0 (0.3)	51.0 (0.3)	78.9 (0.8)	67.4 (0.7)	77.0 (0.2)	94.0 (0.3)	88.9 (0.3)	71.7 (0.2)	8.6
Variants	w/o SK	64.8 (0.8)	34.4 (0.7)	86.7 (3.6)	76.2 (4.0)	77.8 (4.2)	72.4 (2.0)	71.6 (0.4)	50.4 (0.3)	82.7 (0.7)	73.4 (0.7)	79.3 (0.5)	93.9 (0.4)	89.9 (0.4)	71.2 (0.6)	5.5
Variants	w/o Exp	64.6 (1.0)	35.2 (1.1)	85.9 (2.7)	75.4 (4.8)	78.1 (4.4)	72.2 (1.6)	73.9 (0.3)	50.7 (0.1)	81.6 (0.8)	72.1 (0.8)	78.9 (0.4)	94.1 (0.3)	89.7 (0.4)	71.8 (0.2)	6.1
	FUEL	65.2 (0.7)	38.2 (0.8)	87.6 (4.1)	78.1 (0.4)	84.6 (5.0)	73.0 (1.7)	74.6 (0.5)	51.1 (0.7)	83.8 (0.3)	74.1 (0.5)	79.6 (0.6)	94.2 (0.3)	90.1 (0.4)	72.0 (0.1)	2.4

Table 1: Node classification performance. Mean and standard deviation of test accuracy values ($\times 100$) in the node classification task are reported. The best and second-best performances are highlighted in green and yellow colors, respectively. A.R. and O.O.M. denote average ranking and out-of-GPU memory, respectively. 0.0* indicates a value smaller than 10^{-4}. FUEL achieves the best average ranking among the 16 compared methods.

as the loss function. The MLP classifier is then used to measure test accuracy on node classification tasks. Further results using (1) linear classifiers and (2) noisy features are in Appendices D.7 and D.13 of (Kim et al. 2026), respectively.

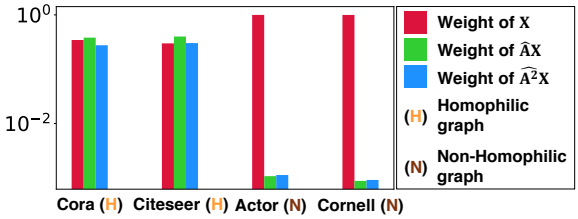
Results. As shown in Table 1, FUEL achieves the best average ranking among 16 methods, demonstrating its effectiveness in learning representations for node classification. Notably, FUEL performs overall best on graphs with diverse graph-level homophily, achieving the best performance on both the least homophilic graph (Texas) and the most homophilic graph (Photo).

5.3 RQ2. Clustering Results

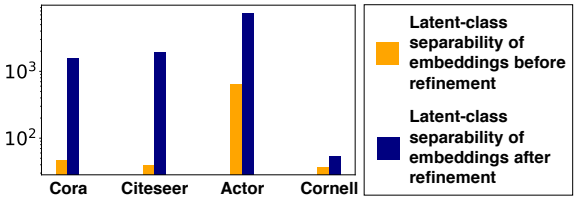
In this section, we evaluate the effectiveness of each method in the node clustering task.

Settings. For each unsupervised node representation learning method, we cluster the learned embeddings using K-Means, setting K to the number of unique labels in the dataset. We then compute the Normalized Mutual Information (NMI) between the resulting clusters and the ground-truth node labels, and regard this NMI score as the measure of clustering performance, following Wang et al. (2024). Additional results using the Adjusted Rand Index (ARI) metric are presented in Section D.8 of the supplementary material.

Results. As shown in Table 2, FUEL achieves the best average ranking among 16 methods, demonstrating that FUEL's effectiveness extends beyond node classification to clustering. Consistent with its performance in node classification, FUEL achieves the best results on both the most and least homophilic graphs, further confirming its adaptability across diverse homophily levels in clustering tasks.



(a) Learned degree of graph convolution usage in Step 1 of FUEL (i.e., weights within the intermediate embeddings).



(b) Latent-class separability of the intermediate embeddings (before refinement) and the final embeddings (after refinement).

Figure 3: Achievement of the design goals of FUEL. (a) demonstrates that FUEL adaptively learns the degree of graph convolution usage according to the level of graph homophily. (b) demonstrates that the refinement step of FUEL enhances the latent-class separability of embeddings.

5.4 RQ3. Achievement of Design Goals

In this section, we evaluate whether FUEL meets its design objectives on real-world graphs.

Settings. We aim to evaluate whether FUEL (1) adaptively learns the degree of graph convolution usage (Section 4.1)

	Datasets	Squirrel	Actor	Wis.	Cornell	Texas	Cham.	Penn94	Flickr	Cora	Citeseer	Pubmed	Photo	Comp.	Arxiv	A.R.
	Homophily	0.217	0.220	0.155	0.111	0.057	0.0	0.483	0.322	0.825	0.717	0.792	0.849	0.802	0.635	-
	Naive $\mathbf{X}$	0.0 (0.0)	5.4 (0.4)	32.8 (2.4)	9.2 (0.8)	17.1 (6.6)	9.1 (2.6)	1.8 (0.9)	1.0 (0.0*)	18.6 (3.4)	20.6 (2.6)	31.0 (0.0*)	13.6 (1.4)	12.3 (1.2)	21.9 (0.1)	12.9
CL	DGI	6.8 (0.2)	0.1 (0.0*)	14.3 (2.3)	11.9 (1.6)	18.3 (2.0)	14.3 (1.0)	1.8 (0.0*)	6.7 (0.5)	54.4 (1.1)	38.6 (1.4)	16.5 (1.4)	48.2 (2.2)	43.2 (1.7)	35.3 (0.4)	11.3
CL	GraphCL	6.6 (0.2)	0.5 (0.0)	13.1 (2.2)	10.5 (2.3)	16.1 (2.3)	13.7 (0.6)	O.O.M.	O.O.M.	58.2 (1.5)	43.3 (0.9)	33.5 (1.1)	53.7 (2.8)	47.5 (1.9)	O.O.M.	10.6
CL	MVGRL	6.0 (0.4)	0.1 (0.0*)	13.5 (1.9)	7.0 (1.7)	22.3 (1.1)	12.5 (0.6)	O.O.M.	6.8 (0.5)	58.1 (1.2)	44.2 (0.8)	33.8 (1.0)	46.5 (3.0)	40.3 (2.7)	O.O.M.	10.9
CL	BGRL	5.5 (0.8)	1.0 (0.4)	12.3 (2.5)	10.0 (1.2)	15.8 (0.9)	14.7 (0.5)	1.4 (0.3)	7.2 (0.3)	55.0 (3.2)	43.2 (1.0)	32.5 (1.2)	51.8 (3.4)	40.0 (1.0)	34.3 (1.1)	11.3
Gen	GAE	7.2 (0.2)	1.5 (0.1)	8.1 (0.8)	11.6 (0.8)	17.1 (3.3)	14.4 (0.8)	O.O.M.	O.O.M.	45.5 (4.9)	33.8 (4.2)	28.5 (1.1)	45.1 (1.0)	38.0 (0.5)	O.O.M.	13.2
Gen	GraphMAE	6.8 (0.3)	1.4 (0.0)	16.7 (1.1)	16.5 (1.0)	19.7 (2.0)	14.9 (0.6)	1.7 (0.0*)	4.7 (0.1)	56.5 (1.2)	44.3 (0.9)	25.7 (1.0)	53.4 (4.1)	32.8 (6.9)	22.2 (1.2)	9.2
Gen	MaskGAE	7.2 (0.7)	1.2 (0.1)	10.9 (2.1)	7.0 (1.1)	19.4 (2.2)	14.7 (0.5)	0.7 (0.1)	5.5 (0.5)	55.5 (2.8)	42.9 (0.6)	14.3 (5.4)	51.6 (2.7)	33.4 (4.1)	36.4 (0.5)	11.8
Non-homophilic	HGRL	8.2 (0.1)	5.9 (0.5)	36.2 (1.6)	40.1 (3.9)	38.5 (3.3)	21.9 (0.4)	O.O.M.	O.O.M.	51.3 (1.8)	42.4 (1.5)	25.2 (2.7)	50.2 (2.0)	46.8 (2.0)	O.O.M.	8.5
Non-homophilic	DSSL	6.6 (0.7)	1.8 (0.2)	12.1 (1.9)	10.5 (1.4)	16.6 (2.2)	15.2 (0.4)	O.O.M.	O.O.M.	54.3 (2.6)	33.7 (1.7)	15.6 (4.4)	69.9 (0.9)	47.2 (2.8)	O.O.M.	12.2
Non-homophilic	GREET	6.2 (0.3)	5.9 (0.5)	43.7 (2.8)	40.0 (2.0)	37.5 (2.0)	21.6 (0.5)	O.O.M.	O.O.M.	57.3 (1.7)	43.4 (1.8)	24.8 (0.6)	55.5 (1.4)	41.1 (1.3)	O.O.M.	8.1
Non-homophilic	NeCo	5.2 (0.4)	0.3 (0.0*)	12.7 (1.1)	17.9 (0.3)	17.9 (1.4)	13.2 (1.6)	O.O.M.	O.O.M.	34.0 (1.0)	16.9 (2.8)	32.1 (1.4)	39.1 (1.6)	32.5 (1.8)	O.O.M.	14.7
Non-homophilic	HLCL	6.3 (0.3)	0.6 (0.2)	12.8 (0.2)	10.4 (4.8)	17.9 (2.2)	14.4 (1.2)	O.O.M.	O.O.M.	55.6 (1.7)	40.4 (0.7)	16.8 (0.2)	55.7 (2.9)	46.1 (0.5)	O.O.M.	12.4
Non-homophilic	PolyGCL	7.8 (0.4)	4.3 (0.7)	44.8 (3.0)	19.8 (5.4)	34.9 (3.3)	19.4 (0.5)	O.O.M.	O.O.M.	35.8 (3.6)	12.4 (2.8)	28.0 (0.9)	42.2 (1.5)	34.0 (3.9)	O.O.M.	10.7
Non-homophilic	HeterGCL	5.4 (0.1)	3.3 (1.1)	27.0 (4.8)	18.1 (2.2)	23.1 (1.2)	15.3 (4.2)	O.O.M.	O.O.M.	51.0 (2.2)	41.5 (1.1)	32.3 (0.3)	56.7 (2.6)	47.4 (1.2)	O.O.M.	9.6
Variants	w/o Step 1	4.4 (0.1)	0.8 (0.1)	17.4 (2.9)	12.4 (0.5)	14.3 (1.7)	14.1 (1.0)	0.9 (0.0*)	2.4 (0.0*)	55.9 (0.9)	43.0 (1.9)	31.7 (0.6)	71.0 (0.9)	53.2 (0.2)	38.2 (0.9)	9.5
Variants	w/o Step 2	5.6 (0.5)	5.4 (0.1)	43.4 (2.8)	35.1 (3.3)	34.0 (3.6)	9.5 (1.2)	0.8 (0.0*)	2.3 (0.2)	48.4 (0.1)	40.9 (1.8)	30.6 (0.1)	66.2 (0.9)	47.4 (1.5)	37.1 (0.1)	9.3
Variants	w/o SK	8.3 (0.3)	5.7 (0.4)	47.7 (0.5)	51.0 (0.1)	39.5 (2.0)	17.3 (1.4)	1.4 (0.1)	2.0 (0.1)	52.6 (0.9)	44.6 (0.1)	26.0 (2.1)	71.1 (0.1)	53.5 (0.3)	39.6 (0.3)	4.9
Variants	w/o Exp	8.8 (0.4)	1.6 (0.6)	45.6 (1.3)	43.5 (0.4)	40.4 (0.3)	10.4 (0.1)	1.2 (0.0*)	1.9 (0.1)	56.1 (0.6)	42.5 (0.9)	29.1 (2.5)	70.9 (0.3)	53.2 (0.5)	40.0 (0.1)	6.4
	FUEL	9.5 (0.2)	6.7 (0.1)	50.1 (0.9)	43.8 (0.6)	40.8 (2.0)	16.8 (1.4)	1.9 (0.0*)	2.2 (0.0*)	59.0 (0.3)	46.1 (0.3)	30.7 (0.4)	71.3 (0.2)	54.4 (0.3)	40.3 (0.1)	2.4

Table 2: Clustering performance. Mean and standard deviation of NMI (Normalized Mutual Information) values between the estimated clusters and the ground-truth clusters ($\times 100$) are reported. The best and second-best performances are highlighted in green and yellow. A.R. and O.O.M. denote average ranking and out-of-GPU memory, respectively. 0.0* indicates a value smaller than 10^{-4}. FUEL achieves the best average ranking among the 16 compared methods.

and (2) enhances the latent-class separability of embeddings through refinement (Section 4.2). For the first goal, we analyze the learned weights in the intermediate embeddings $\mathbf{H}$ (Eq. (6)): the weight of $\mathbf{X}$ (α_0^*), the weight of $\hat{\mathbf{A}}\mathbf{X}$ (α_1^*), and the weight of $\hat{\mathbf{A}}^{\mathbf{2}}\mathbf{X}$ (α_2^*). This analysis is conducted on both homophilic graphs (Cora and Citeseer) and non-homophilic graphs (Actor and Cornell) to verify whether each weight is adaptively learned based on the respective graph's homophily level. For the second goal, we compare the latent-class separability of the intermediate embeddings (before refinement), and the final embeddings (after refinement) using the Calinski–Harabasz index, as in Section 3.1.

Results. First, as shown in Figure 3 (a), FUEL adaptively adjusts the degree of graph convolution based on graph homophily. Specifically, in homophilic graphs (Cora and Citeseer), FUEL assigns a significantly higher sum of weights to graph-convolution-related terms ($\hat{\mathbf{A}}\mathbf{X}$ and $\hat{\mathbf{A}}^{\mathbf{2}}\mathbf{X}$) compared to the node features $\mathbf{X}$. In contrast, in non-homophilic graphs (Actor and Cornell), the weights assigned to graph-convolution-related terms are reduced to near zero. Second, as shown in Figure 3 (b), FUEL significantly enhances the latent-class separability of the embeddings through its refinement step. These results demonstrate that FUEL successfully achieves its design objectives on real-world graphs with various homophily levels.

5.5 RQ4. Ablation Study

In this section, we assess whether each key component of FUEL is essential for achieving high performance.

Settings. We use 4 variants: (1) FUEL that does not use the specialized clustering scheme (w/o Step 1), (2) FUEL that does not perform the refinement step (w/o Step 2), (3) FUEL that does not use skip connection in the refiner (w/o SK), and (4) FUEL that does not use an exponential function for the loss (Eq. (7)) (w/o Exp). Details on the variants are provided in Section C.3 of the supplementary material. We evaluate each variant in both node classification and clustering, under the same setting of Section 5.2 and 5.3, respectively.

Results. As shown in Tables 1 and 2, FUEL outperforms its variants in 10 out of 14 datasets for both node classification and clustering tasks (see rows corresponding to **Variants**). This result highlights the necessity of FUEL's key modules in obtaining high-quality node embeddings.

6 Conclusion

In this work, we investigate unsupervised node representation learning without the homophily assumption. We begin by providing both theoretical and empirical analyses that highlight the necessity of adjusting the degree of graph convolution usage and suggesting an unsupervised method for this adjustment (Section 3). Building on these insights, we introduce FUEL, a feature-centric approach that adaptively learns the adequate degree of graph convolution usage (Section 4). Through extensive experiments including 15 baseline methods and 14 benchmark datasets, we demonstrate the effectiveness of FUEL in both node classification and clustering tasks (Section 5). As future work, we believe our approach can be applied to learning node representations on more diverse graph types, including heterogeneous (Wang et al. 2022; Zhao et al. 2021) and text-attributed graphs (Yan et al. 2023; Kim et al. 2025). For reproducibility, code and datasets are in https://github.com/kswoo97/unsupervised-non-homophilic.

Ethical Statement

This work proposes an unsupervised technique for learning node representations. The experiments are conducted on publicly available benchmark datasets that have been widely used in prior work. We do not collect new human subject data, nor do we attempt to infer sensitive attributes. While we believe our experiments do not involve any harmful or toxic content, applying our method in sensitive domains (e.g., social or financial networks) requires careful use to avoid potential privacy violations or unintended negative impacts.

Acknowledgements

This work was partly supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2024-00406985, 40%). This work was partly supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2022-II220871, Development of AI Autonomy and Knowledge Enhancement for AI Agent Collaboration, 50%) (No. RS-2019-II190075, Artificial Intelligence Graduate School Program (KAIST), 10%).

References

Asano, Y. M.; Rupprecht, C.; and Vedaldi, A. 2020. Self-labelling via simultaneous clustering and representation learning. In *ICLR*.

Bishop, C. M.; and Nasrabadi, N. M. 2006. *Pattern recognition and machine learning*, volume 4. Springer.

Caliński, T.; and Harabasz, J. 1974. A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods*, 3(1): 1–27.

Caron, M.; Misra, I.; Mairal, J.; Goyal, P.; Bojanowski, P.; and Joulin, A. 2020. Unsupervised learning of visual features by contrasting cluster assignments. In *NeurIPS*.

Chen, J.; Lei, R.; and Wei, Z. 2024. PolyGCL: GRAPH CONTRASTIVE LEARNING via Learnable Spectral Polynomial Filters. In *ICLR*.

Chen, J.; Zhu, G.; Qi, Y.; Yuan, C.; and Huang, Y. 2022. Towards self-supervised learning on graphs with heterophily. In *CIKM*.

Chien, E.; Peng, J.; Li, P.; and Milenkovic, O. 2021. Adaptive universal generalized pagerank graph neural network. In *ICLR*.

Fey, M.; and Lenssen, J. E. 2019. Fast graph representation learning with PyTorch Geometric. In *ICLR workshop on representation learning on graphs and manifolds*.

Gasteiger, J.; Bojchevski, A.; and Günnemann, S. 2019. Predict then propagate: Graph neural networks meet personalized pagerank. In *ICLR*.

Ghasedi Dizaji, K.; Herandi, A.; Deng, C.; Cai, W.; and Huang, H. 2017. Deep clustering via joint convolutional autoencoder embedding and relative entropy minimization. In *CVPR*.

Grover, A.; and Leskovec, J. 2016. node2vec: Scalable feature learning for networks. In *KDD*.

Guo, X.; Wang, Y.; Wei, Z.; and Wang, Y. 2023. Architecture matters: Uncovering implicit mechanisms in graph contrastive learning. In *NeurIPS*.

Hassani, K.; and Khasahmadi, A. H. 2020. Contrastive multi-view representation learning on graphs. In *ICML*.

He, D.; Zhao, J.; Guo, R.; Feng, Z.; Jin, D.; Huang, Y.; Wang, Z.; and Zhang, W. 2023. Contrastive learning meets homophily: two birds with one stone. In *ICML*.

Hou, Z.; He, Y.; Cen, Y.; Liu, X.; Dong, Y.; Kharlamov, E.; and Tang, J. 2023. Graphmae2: A decoding-enhanced masked self-supervised graph learner. In *WWW*.

Hou, Z.; Liu, X.; Cen, Y.; Dong, Y.; Yang, H.; Wang, C.; and Tang, J. 2022. Graphmae: Self-supervised masked graph autoencoders. In *KDD*.

Huang, J.; Gong, S.; and Zhu, X. 2020. Deep semantic clustering by partition confidence maximisation. In *CVPR*.

Ju, M.; Zhao, T.; Wen, Q.; Yu, W.; Shah, N.; Ye, Y.; and Zhang, C. 2023. Multi-task self-supervised graph neural networks enable stronger task generalization. In *ICLR*.

Khorasgani, S. H.; Chen, Y.; and Shkurti, F. 2022. Slic: Self-supervised learning with iterative clustering for human action videos. In *CVPR*.

Kim, S.; Kang, S.; Bu, F.; Lee, S. Y.; Yoo, J.; and Shin, K. 2024. Hypeboy: Generative self-supervised representation learning on hypergraphs. In *ICLR*.

Kim, S.; Lee, S. Y.; Kim, K.; Hwang, H.; Yoo, J.; and Shin, K. 2026. Supplementary materials for this work. https://github.com/kswoo97/unsupervised-non-homophilic.

Kim, S.; Lee, S. Y.; Yoo, J.; and Shin, K. 2025. 'Hello, World!': Making GNNs Talk with LLMs. In *EMNLP Findings*.

Kipf, T. N.; and Welling, M. 2016. Variational graph auto-encoders. In *NeurIPS workshop on bayesian deep learning*.

Kipf, T. N.; and Welling, M. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.

Lee, S. Y.; Kim, S.; Bu, F.; Yoo, J.; Tang, J.; and Shin, K. 2024. Feature distribution on graph topology mediates the effect of graph convolution: Homophily perspective. In *ICML*.

Li, J.; Wu, R.; Sun, W.; Chen, L.; Tian, S.; Zhu, L.; Meng, C.; Zheng, Z.; and Wang, W. 2023. What's Behind the Mask: Understanding Masked Graph Modeling for Graph Autoencoders. In *KDD*.

Liu, Y.; Zheng, Y.; Zhang, D.; Lee, V. C.; and Pan, S. 2023. Beyond smoothing: Unsupervised graph representation learning with edge heterophily discriminating. In *AAAI*.

Luan, S.; Hua, C.; Lu, Q.; Zhu, J.; Zhao, M.; Zhang, S.; Chang, X.-W.; and Precup, D. 2022. Revisiting heterophily for graph neural networks. In *NeurIPS*.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. Deepwalk: Online learning of social representations. In *KDD*.

Thakoor, S.; Tallec, C.; Azar, M. G.; Munos, R.; Veličković, P.; and Valko, M. 2021. Bootstrapped representation learning on graphs. In *ICLR workshop on geometrical and topological representation learning*.

Veličković, P.; Fedus, W.; Hamilton, W. L.; Liò, P.; Bengio, Y.; and Hjelm, R. D. 2019. Deep graph infomax. In *ICLR*.

Walawalkar, D.; and Garrido, P. 2024. VideoClusterNet: Self-supervised and Adaptive Face Clustering for Videos. In *ECCV*.

Wang, C.; Liu, Y.; Yang, Y.; and Li, W. 2024. HeterGCL: Graph Contrastive Learning Framework on Heterophilic Graph. In *IJCAI*.

Wang, X.; Bo, D.; Shi, C.; Fan, S.; Ye, Y.; and Yu, P. S. 2022. A survey on heterogeneous graph embedding: methods, techniques, applications and sources. *IEEE transactions on big data*, 9(2): 415–436.

Wu, F.; Souza, A.; Zhang, T.; Fifty, C.; Yu, T.; and Weinberger, K. 2019. Simplifying graph convolutional networks. In *ICML*.

Xiao, T.; Chen, Z.; Guo, Z.; Zhuang, Z.; and Wang, S. 2022. Decoupled self-supervised learning for graphs. In *NeurIPS*.

Yan, H.; Li, C.; Long, R.; Yan, C.; Zhao, J.; Zhuang, W.; Yin, J.; Zhang, P.; Han, W.; Sun, H.; et al. 2023. A comprehensive study on text-attributed graphs: Benchmarking and rethinking. In *NeurIPS*.

Yang, W.; and Mirzasoleiman, B. 2024. Graph Contrastive Learning under Heterophily via Graph Filters. In *UAI*.

You, Y.; Chen, T.; Sui, Y.; Chen, T.; Wang, Z.; and Shen, Y. 2020. Graph contrastive learning with augmentations. In *NeurIPS*.

Yu, Z.; Feng, B.; He, D.; Wang, Z.; Huang, Y.; and Feng, Z. 2024. LG-GNN: local-global adaptive graph neural network for modeling both homophily and heterophily. In *IJCAI*.

Zhao, B.; and Mac Aodha, O. 2023. Incremental generalized category discovery. In *ICCV*.

Zhao, J.; Wang, X.; Shi, C.; Hu, B.; Song, G.; and Ye, Y. 2021. Heterogeneous graph structure learning for graph neural networks. In *AAAI*.

Zheng, X.; Wang, Y.; Liu, Y.; Li, M.; Zhang, M.; Jin, D.; Yu, P. S.; and Pan, S. 2022. Graph neural networks for graphs with heterophily: A survey. *arXiv preprint arXiv:2202.07082*.

Zhu, J.; Yan, Y.; Zhao, L.; Heimann, M.; Akoglu, L.; and Koutra, D. 2020. Beyond homophily in graph neural networks: Current limitations and effective designs. In *NeurIPS*.