UNIFEWS: You Need Fewer Operations for Efficient Graph Neural Networks

Ningyi Liao, Zihao Yu, Ruixiao Zeng, Siqiang Luo
Nanyang Technological University, Singapore
{liao0090, zihao010, zeng0146}@e.ntu.edu.sg, siqiang.luo@ntu.edu.sg

***Abstract*—Graph Neural Networks (GNNs) have shown promising performance in various graph learning tasks, but at the cost of resource-intensive computations. The primary overhead of GNN update stems from graph propagation and weight transformation, both involving operations on graph-scale matrices. While previous studies attempt to reduce the computational budget, these methods mostly leverage graph-level or network-level sparsification techniques with limited flexibility. In this work, we propose UNIFEWS, which unifies the two operations in an entry-wise manner considering individual matrix elements, and conducts joint edge-weight sparsification to enhance learning efficiency. The UNIFEWS design enables adaptive compression across GNN layers with progressively increased sparsity, and is applicable to a variety of architectural designs with on-the-fly operation simplification. Theoretically, we establish a novel framework to characterize sparsified GNN learning in view of the graph optimization process, and prove that UNIFEWS effectively approximates the learning objective with bounded error and reduced computational overhead. We conduct extensive experiments to evaluate the performance of our method in diverse settings. UNIFEWS is advantageous in jointly removing more than 90% of edges and weight entries with comparable or better accuracy than baseline models. The sparsification offers remarkable efficiency improvements including $10-20\times$ matrix operation reduction and up to $100\times$ acceleration in graph propagation time for the largest graph at the billion-edge scale. Our code is available at: https://github.com/nyLiao/Unifews.**

***Index Terms*—Graph Neural Networks, Acceleration, Graph Spectral Theory, Graph Sparsification**

I. INTRODUCTION

Graph Neural Networks (GNNs) have undergone extensive development for learning from graph-structure data and have achieved remarkable performance on a variety of tasks [1]–[4]. The power of GNNs is mainly attributed to their message-passing scheme that each node in the graph aggregates information from its neighbors, namely *graph propagation*, and then updates its own representation accordingly by trainable weights as *feature transformation*. Iteratively performing the two-stage scheme enables the model to retrieve information and learn from the graph topology inclusively.

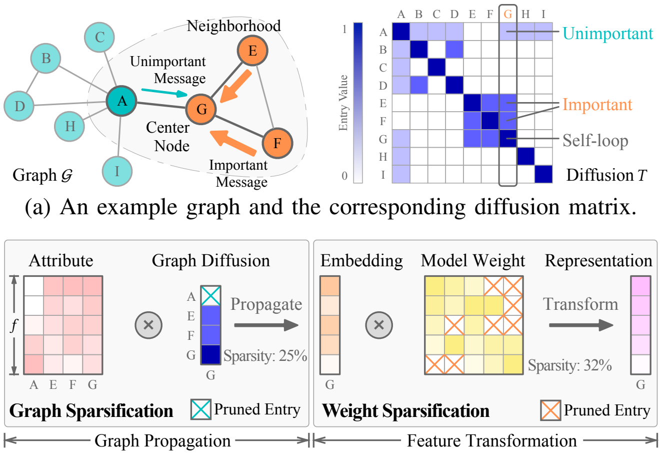
Despite their success, canonical GNNs are recognized for their high resource consumption, especially when scaling up to large graphs [5]. The performance bottleneck arises from the sequence of matrix multiplications integral to message passing stages, i.e., graph propagation and feature transformation, which alternatively apply to the node representation and lead to computational complexities directly proportional to the numbers of edges and nodes in the graph, respectively [6]. Hence, the essence of improving GNN learning efficiency lies in reducing the number of operations associated with graph diffusion and model weights [7], [8].

Motivation: Improve Efficiency through Graph-Weight Co-design. Empirically, a diverse range of strategies have been explored for saving computational cost by sparsifying the graph as well as the model. In efforts to simplify graph computation, prior research utilizes graph sparsification techniques [9]. This typically entails removing graph components based on either predetermined [10]–[12] or learned [13]–[15] criteria. However, the persistence of a static graph through all propagation hops causes a dilemma: the topology may become overly coarse, thereby omitting crucial information for GNN feature extraction, or the graph structure may be under-sparsified and model efficiency is scarcely improved.

Another direction is to exploit compression on the model architecture by integrating neural network pruning approaches [16], which gradually sparsifies weight elements during GNN training based on performance assessments [17]–[20]. Regardless of the improved flexibility, their scalability for large graphs remains limited, as full-graph computation is still necessary at certain stages of the pipeline. Moreover, there is a noticeable gap in the theoretical interpretation of GNN sparsification despite its practical utility. Existing works are either constrained to the layer-agnostic graph approximation [21], [22], or only provide straightforward representation error analysis [12], [23]. The impact of simplified graphs on the GNN learning process, particularly through multiple layers, remains inadequately addressed.

To mitigate the above drawbacks of current GNN compression solutions, we propose to rethink both graph propagation and feature transformation from an entry-wise perspective, i.e., examining individual matrix elements. As illustrated in Fig. 1, the two operations are performed by employing matrix multiplications with the diffusion and weight matrices, respectively. Hence, directly sparsifying entries in these two matrices enables us to reduce the number of *entry operations* in a unified fashion. The entry-wise mechanism enjoys fine-grained flexibility compared to conventional graph-level sparsification techniques, while ensuring a deterministic graph topology for network learning with reasonable overhead.

Our Contribution. In this paper, we propose **UNIFEWS**, a UNIFied Entry-WIse Sparsi-fication framework for GNN graph and weights. Theoretically, we establish a quantitative framework featuring UNIFEWS with respect to the graph



(a) An example graph and the corresponding diffusion matrix.

(b) UniFEWS sparsification on graph and model entries.

Fig. 1: **(a)** In graph propagation, messages from neighbors with larger diffusion values are associated with greater importance. **(b)** UniFEWS jointly applies sparsification to both GNN graph propagation and feature transformation stages. Color intensity of an entry denotes its relative magnitude. Unimportant entries of the diffusion and weight matrices are pruned in order to reduce computational operations.

learning objective, bridging the spectral filtering explanation of GNN learning and the sparsification procedure with a clear precision bound. Different from previous approaches focusing on a particularized metric for approximation, our analysis elucidates the effect of joint sparsification across iterative GNN layers in a holistic manner.

In practice, UniFEWS is capable of jointly removing entry operations in both graph propagation and feature transformation towards improved model efficiency. The simple but effective strategy is adeptly integrated into matrix computations on the fly without incurring additional overhead. The progressive sparsification across layers further fosters an increase in sparsity for both stages, which reciprocally augments their efficiency. Consequently, UniFEWS is powerful in conducting personalized propagation as well as transformation for each node and each layer, effectively balancing the operational reduction while retaining precision. The theoretical and implementation framework of UniFEWS is applicable to a broad family of message-passing GNNs, covering the representative models of both iterative and decoupled architectures.

Our main contributions are outlined as the following:

- **New Methodology:** We propose UniFEWS as a unified GNN sparsification, which employs an entry-wise scheme on both graph connections and model weights. UniFEWS is adaptable to common GNN designs to reduce graph-scale operations and alleviate computational cost.
- **New Theory:** We develop novel theoretical analysis and guarantee of UniFEWS approximation through the comprehensive lens of GNN learning process. UniFEWS enhances sparsity by adaptively simplifying and accumulating operations throughout network layers.
- **Superior Compression:** We experimentally evaluate UniFEWS with a wide range of GNN architectures and graph datasets. Our method outperforms state-of-the-art GNN compressions by reaching over 90% of joint sparsity without compromising accuracy. On the billion-scale graph papers100M, UniFEWS is able to boost the computation time by 100×.

Organization. In the remainder of this paper, Section II outlines the related approaches for enhancing GNN efficiency, while Section III derives the theoretical framework for characterizing the GNN learning process in view of spectral graph smoothing. Section IV develops the UniFEWS design for decoupled and joint sparsifications successively, together with respective algorithms and precision guarantees. In Section V, extensive experiments are conducted to study the performance of UniFEWS under various model architectures, sparsification schemes, and parameter configurations.

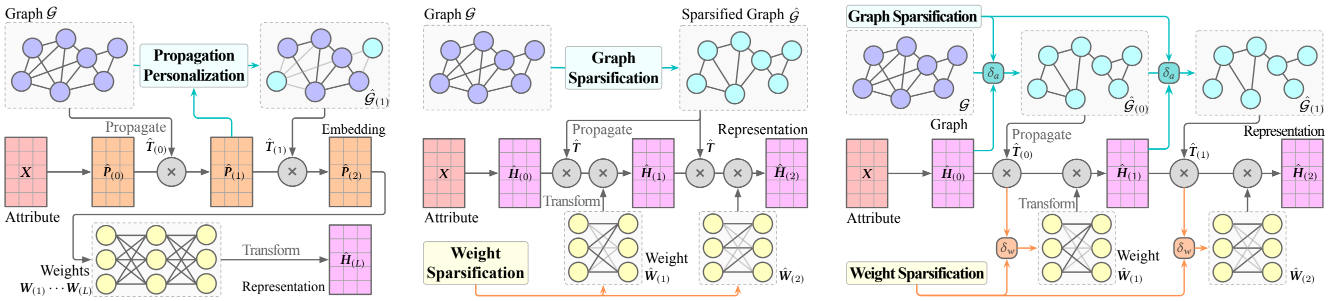
II. Related Work

Iterative Graph Sparsification. Graph sparsification methods mainly modify graph edges with a set of graph management techniques, mostly for the *iterative* message-passing scheme. As the main focus is data augmentation, these methods ensure the expressivity and compatibility of GNN architectures [7]. Among them, a large portion of works utilize graph sparsification for accuracy improvement, typically through deleting edges. NeuralSparse [10] and LSP [24] drop edges based on k-neighbor metric for each node and layer-wise local graph topology, respectively, while AdaptiveGCN [13] and SGCN [14] choose to implement sparsity by learnable edge features and graph adjacency.

Specifically, a few works examine *spectral sparsification* as an attempt to boost efficiency. FastGAT [23] eliminates connections in the GAT [3] model by calculating the Effective Resistance [25] but at the expense of high computational time. DSpar [12] employs a loose approximation and successfully applies to larger datasets. Other graph management techniques such as graph coarsening [15], [26]–[28] are also utilized for simplifying the GNN computation. Most of the simplification techniques, however, conduct graph modification in an one-time and layer-agnostic manner, resulting in limited flexibility and utility under the trade-off between efficiency and efficacy.

Decoupled Propagation Personalization. A particularized branch for simplifying *decoupled* GNNs recently emerges as propagation personalization. Since the graph operation is isolated from model updates, it is more flexible for fine-grained data modification. Fig. 2(a) illustrates their idea for replacing the graph-level message-passing with separate maneuver on the graph connection for each propagation hop. NDLS [29] and NIGCN [30] personalize the hop number per node in the decoupled GNN design as an approach to acknowledge the local structure and mitigate over-smoothing. SCARA [31], [32] replaces the hop-base message-passing with a graph-oriented pushing scheme and eager termination, while Shadow-GNN [11] explicitly selects the appropriate node-wise aggregation neighborhood.

Another set of works [33]–[36] achieves customized diffusion by gradually learning the hop-level importance of nodes at



(a) Propagation Personalization (b) Joint Compression (c) UNIFEWS (on Iterative GNN)

Fig. 2: Comparison of GNN learning pipelines between conventional simplification techniques and our UNIFEWS framework. **(a)** The approach of personalized propagation iteratively simplifies the node-dependent graph diffusion but is only applicable to decoupled GNNs. **(b)** Joint model compression can sparsify both graph and weight, whereas the same diffusion is uniformly utilized across all layers. **(c)** In contrast, our UNIFEWS framework implements entry-wise sparsification of both graph and weights for each layer. The progressive scheme enjoys fine-grained control and improved sparsity.

the price of introducing more overhead, which are less relevant to the efficient GNN computation. These personalizations can be equivalently regarded as modifications on graphs with more flexible and dedicated arrangements. Our UNIFEWS is akin to this idea for tuning the message-passing process, but with a more detailed entry-wise scheme and a better suitability for both iterative and decoupled propagations.

Architectural and Joint Compression. The concept of GNN *pruning* takes the neural network architecture into account and exploits sparsification for efficiency without hindering effectiveness [37]. [17] consider structural channel pruning by reducing the dimension of weight matrices, while CGP [38] appends a prune-and-regrow scheme on the graph structure.

A line of research refers to the *lottery ticket hypothesis* [39], [40] in the context of graph learning in search of a smaller yet effective substructure in the network. Their general pipeline, as outlined in Fig. 2(b), involves a model compressor and a static graph sparsifier. Among these approaches, GEBT [19] finds the initial subnetwork with simple magnitude-based pruning on both adjacency and weight matrices, whilst ICPG [41] learns the importance of edges as an external task. GLT [18] and DGLT [20] employ an alternative optimization strategy that progressively processes the graph and network components. Other compression techniques such as *quantization* [42]–[44] are also investigated. One critical drawback of these methods lies in the additional bottleneck for searching or learning the compression strategy, which still incurs full-scale computation and are less efficient.

Graph Sampling. Sampling is another common strategy for improving GNN efficiency by altering the message-passing process. It is based on the intuition that the learning process only require a portion of data samples for each iteration. Thence, graph-, layer-, and node-level strategies are designed for retrieving useful information [21], [22], [45]–[48]. A contemporary work [49] also investigates the utilization of spectral sparsification for sampling decoupled GNNs in the polynomial form, which echos with our theoretical analysis that graph sparsification process is useful in offering bounded approximation for message-passing GNNs. We however note that the sampling approach is limited as it is derived from the iterative scheme and does not produce a deterministic sparsified graph. Consequently, it is not applicable to decoupled models which is dominant on large graphs. Generally, GNNs with graph sampling are considered as individual models, which are less relevant to this study for proposing a unified augmentation across various models and schemes.

III. GNN AND GRAPH SMOOTHING

In this section, we theoretically characterize general GNN learning by utilizing its spectral explanation, with preliminary concepts and the graph smoothing framework introduced in Sections III-A and III-B, respectively. Then, we develop the approximate graph smoothing process under graph modification in Section III-C in preparation of GNN sparsification.

A. GNN Architecture

Graph Notation. Consider a graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ with $n = |\mathcal{V}|$ nodes and $m = |\mathcal{E}|$ edges. The neighborhood set of a node $u \in \mathcal{V}$ is $\mathcal{N}(u) = \{v | (u, v) \in \mathcal{E}\}$, and its degree $d(u) = |\mathcal{N}(u)|$. The self-looped [1] adjacency matrix is $\bar{\boldsymbol{A}} \in \mathbb{R}^{n\times n}$ and the diagonal degree matrix is $\boldsymbol{D} = \operatorname{diag}(d(u_1), d(u_2), \cdots, d(u_n))$. The normalized adjacency and Laplacian matrices are $\tilde{\boldsymbol{A}} = \boldsymbol{D}^{-1/2}\bar{\boldsymbol{A}}\boldsymbol{D}^{-1/2}$ and $\tilde{\boldsymbol{L}} = \boldsymbol{I} - \tilde{\boldsymbol{A}}$, respectively. Summary for primary symbols and notations are in Table I.

Iterative GNN. Iterative models such as GCN [1] take the node attribute $\boldsymbol{X} \in \mathbb{R}^{n\times f}$ as input, and recurrently update by applying the graph diffusion matrix $\boldsymbol{T}$. The representation matrix $\boldsymbol{H}_{(l+1)}$ of the $(l+1)$-th layer is thus updated as:

$$\boldsymbol{H}_{(l+1)} = \sigma\left(\boldsymbol{T}\boldsymbol{H}_{(l)}\boldsymbol{W}_{(l)}\right), \quad l = 0, 1, \cdots, L-1, \tag{1}$$

where $\sigma(\cdot)$ denotes the activation function. Eq. (1) implies two consecutive steps, that *graph propagation* computes the embedding $\boldsymbol{P}_{(l)} = \boldsymbol{T}\boldsymbol{H}_{(l)}$, and *feature transformation* multiplies the learnable weight $\boldsymbol{W}_{(l)}$. For GCN, the graph diffusion matrix is $\boldsymbol{T} = \tilde{\boldsymbol{A}}$, whereas in GAT [3], [50], the diffusion matrix $\boldsymbol{T}$ is composed of the attention weights of edges.

Decoupled GNN. This variant simplifies GNN architecture by separating the propagation from iterative updates [51]–[53]. The graph-related propagation operations can be computed in advance as the embedding matrix $\boldsymbol{P}_{(L)}$, and the transformation is as simple as an L-layer MLP. Here we formulate the decoupled GNN with the two consecutive stages as:

$$\boldsymbol{P}_{(l+1)} = \boldsymbol{T}_{(l)} \cdot \boldsymbol{P}_{(l)}, \qquad l = 0, 1, \cdots, L-1, \tag{2}$$

$$\boldsymbol{H}_{(l+1)} = \sigma\left(\boldsymbol{H}_{(l)} \boldsymbol{W}_{(l)}\right), \quad l = 0, 1, \cdots, L-1, \tag{3}$$

where the boundary conditions are $\boldsymbol{P}_{(0)} = \boldsymbol{X}$ and $\boldsymbol{H}_{(0)} = \boldsymbol{P}_{(L)}$. As an exemplar, the diffusion matrices for SGC [51] and APPNP [54] are $\boldsymbol{T} = \tilde{\boldsymbol{A}}$ and $\boldsymbol{T} = \alpha(1-\alpha)\tilde{\boldsymbol{A}}$ according to their respective propagation designs.

B. GNN as Graph Smoothing

A broad scope of GNN models can be explained by a spectral graph smoothing process. We employ the general framework linking both iterative and decoupled GNN learning with a unified optimization objective [55], [56]:

*Definition 1 (**Graph Laplacian Smoothing** [55]):* Given a weighted graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ with Laplacian matrix $\boldsymbol{L}$. Based on an input signal vector $\boldsymbol{x} \in \mathbb{R}^n$, the Graph Laplacian Smoothing problem aims to optimize vector $\boldsymbol{p} \in \mathbb{R}^n$ with the goal:

$$\boldsymbol{p}^* = \arg\min_{\boldsymbol{p}} \mathcal{L}, \quad \mathcal{L} = \|\boldsymbol{p} - \boldsymbol{x}\|^2 + c \cdot \boldsymbol{p}^\top \boldsymbol{L} \boldsymbol{p}, \tag{4}$$

where $\|\cdot\|$ is the vector L_2 norm and the regularization coefficient c is chosen from $[0,1]$.

In Eq. (4), $\boldsymbol{L}$ is the general Laplacian matrix, as normalization only causes a difference in coefficient. The first term of $\mathcal{L}$ reflects the closeness to the *input signal*, i.e., node attributes representing their identity. The second term is associated with the *graph structure*, acting as a regularization that constrains the representation values of connected node pairs to be similar. The closed-form solution $\boldsymbol{p}^* = (\boldsymbol{I} + c\boldsymbol{L})^{-1}\boldsymbol{x}$ can be inferred when the derivative $\partial\mathcal{L}/\partial\boldsymbol{p} = 0$. However, note that it is prohibitive to directly acquire the converged solution due to the inverse calculation on large graph-scale matrix. Hence, GNN models employ an iterative approach to learn the representation under this framework with varying architectural designs. For example, GCN convolution corresponds to $c = 1$, while GAT embraces a variable coefficient $c[u,v]$ related with the attention of each node pair (u,v).

TABLE I: Summary of primary symbols and notations.

Notation	Description
$\mathcal{G}, \mathcal{V}, \mathcal{E}$	Graph, node set, and edge set
$\mathcal{N}(u)$	Neighboring node set of node u
n, m, f	Node, edge, and feature size
L	Number of propagation hops and network layers
c	Graph smoothing regularization coefficient
b	Graph smoothing gradient descent step size
ϵ	Graph spectral approximation rate
δ_a, δ_w	Thresholds for graph and weight sparsification
q_a, q_w	Numbers of removed edges and weight entries
η_a, η_w	Graph sparsity and weight sparsity
$\bar{\boldsymbol{A}}$, $\tilde{\boldsymbol{A}}$	Self-looped and normalized adjacency matrix of graph $\mathcal{G}$
$\bar{\boldsymbol{L}}$, $\tilde{\boldsymbol{L}}$	Raw and normalized Laplacian matrix of graph $\mathcal{G}$
$\hat{\boldsymbol{A}}$, $\hat{\boldsymbol{L}}$	Approximate adjacency and Laplacian matrix of graph $\hat{\mathcal{G}}$
$\boldsymbol{T}$, $\boldsymbol{W}$	Graph diffusion matrix and weight matrix
$\tau[u,v]$	Message entry corresponding to edge (u,v)
$\omega[j,i]$	Network entry corresponding to neuron (j,i)
$\boldsymbol{X}$, $\boldsymbol{x}$	Node attribute matrix and feature-wise vector
$\boldsymbol{P}_{(l)}$, $\boldsymbol{p}_{(l)}$	Embedding matrix and feature-wise vector of layer l
$\boldsymbol{H}_{(l)}$, $\boldsymbol{h}_{(l)}$	Representation matrix and feature-wise vector of layer l

More generally, when there are f input vectors, i.e., the input matrix is $\boldsymbol{X} \in \mathbb{R}^{n \times f}$, then the matrix form of graph Laplacian smoothing corresponding to Eq. (4) is:

$$\boldsymbol{P}^* = \arg\min_{\boldsymbol{P}} \|\boldsymbol{P} - \boldsymbol{X}\|_F^2 + c \cdot \mathrm{tr}(\boldsymbol{P}^\top \boldsymbol{L} \boldsymbol{P}), \tag{5}$$

where $\|\cdot\|_F$ is the matrix Frobenius norm and the closed-form solution is $\boldsymbol{P}^* = (\boldsymbol{I} + c\boldsymbol{L})^{-1}\boldsymbol{X}$.

C. Approximate Graph Smoothing

Graph sparsifiers employed in efficient GNNs modify the graph topology and consequently affect the graph smoothing process throughout GNN learning. In order to measure the extend of the approximation process and its impact on learning outcomes, we introduce the spectral similarity considering a single signal vector:

*Definition 2 (**ϵ-Spectral Similarity**):* The approximate Laplacian matrix $\hat{\boldsymbol{L}}$ is said to be ϵ-spectrally similar to the raw Laplacian matrix $\boldsymbol{L}$ if:

$$\boldsymbol{x}^\top (\hat{\boldsymbol{L}} - \epsilon \boldsymbol{I}) \boldsymbol{x} \leq \boldsymbol{x}^\top \boldsymbol{L} \boldsymbol{x} \leq \boldsymbol{x}^\top (\hat{\boldsymbol{L}} + \epsilon \boldsymbol{I}) \boldsymbol{x}, \quad \forall \boldsymbol{x} \in \mathbb{R}^n. \tag{6}$$

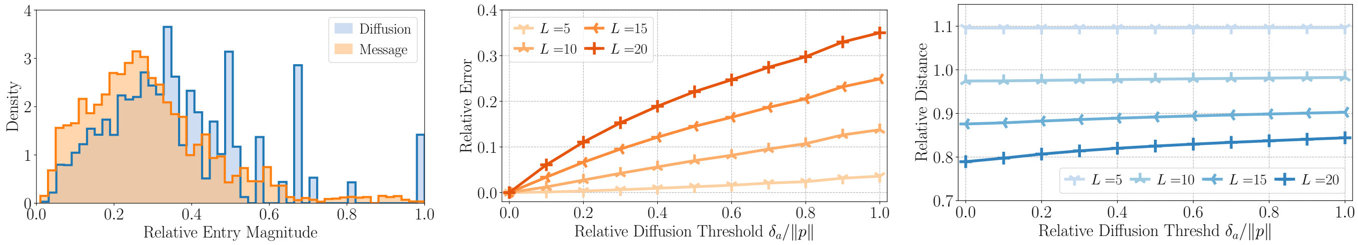
Graph approximation satisfying Definition 2 can be regarded as spectral sparsification [25], [57], which identifies a family of operations maintaining certain spectral properties such as eigenvalues of the graph during changes. Compared to the common *multiplicative* spectral similarity [58]–[60], Definition 2 possesses an *additive* tolerance applied onto the graph Laplacian, which allows for modification on specific entries of the matrix $\boldsymbol{L}$ and suits our scenario.

Then, we are able to characterize the graph smoothing problem for sparsified graphs. Under edge modifications, we define the approximate spectral smoothing process in the following lemma as an approach for bounding the optimization goal under approximation:

*Lemma 1 (**Approximate Graph Laplacian Smoothing**):* Given two graphs $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ and $\hat{\mathcal{G}} = \langle \mathcal{V}, \hat{\mathcal{E}} \rangle$, where $\hat{\mathcal{E}}$ is the sparsified edge set. When the Laplacian $\hat{\boldsymbol{L}}$ of $\hat{\mathcal{G}}$ is ϵ-similar to $\boldsymbol{L}$ of $\mathcal{G}$, the solution $\hat{\boldsymbol{p}}^*$ to the problem Eq. (4) w.r.t $\hat{\boldsymbol{L}}$ is called an ϵ-approximation of the solution $\boldsymbol{p}^*$ w.r.t. $\boldsymbol{L}$, and:

$$\|\hat{\boldsymbol{p}}^* - \boldsymbol{p}^*\| \leq c\epsilon \|\boldsymbol{p}^*\|. \tag{7}$$

Due to page limit, the proof of Lemma 1 can be found in the tech report [61]. It establishes a novel interpretation for characterizing the smoothing procedure under a sparsified graph, that if a sparsifier complies with the spectral similarity Definition 2, it is capable of effectively approximating the iterative graph optimization and achieving a close output with bounded error. Compared to approximation bounds specifying



(a) Distribution of Entries (b) Approximate Embedding Error (c) Distance to Over-smoothed Embedding

Fig. 3: Empirical evaluation on the SGC graph propagation and the effect of UNIFEWS on cora dataset. **(a)** Distribution of entries in the diffusion matrix $\boldsymbol{T}$ and the message $\boldsymbol{\tau}_{(0)}$ at hop $l = 0$. **(b)** The relative error margin $\|\hat{\boldsymbol{P}}_{(L)} - \boldsymbol{P}_{(L)}\|_F / \|\boldsymbol{P}_{(L)}\|_F$ of approximation to the raw embedding against the strength of UNIFEWS and propagation hops. **(c)** The relative distance $\|\hat{\boldsymbol{P}}_{(L)} - \boldsymbol{P}^*\|_F / \|\boldsymbol{P}^*\|_F$ to the converged embedding against the strength of UNIFEWS and propagation hops.

feature values in previous GNN sparsification theories [23], [37], our analysis highlights the impact of graph sparsifier throughout the holistic learning process, which enjoys better expressiveness and suitability.

IV. UNIFEWS: UNIFIED ENTRY-WISE SPARSIFICATION

This section presents the UNIFEWS framework by respectively developing its application to decoupled and iterative GNNs, where graph and weight sparsification are separately performed in the former architecture, and are combined in the latter. The insights of our theoretical analysis in this section are summarized as below:

- **Bridging sparsification and smoothing:** UNIFEWS can be described as a graph spectral sparsifier on both decoupled and iterative GNN architectures. Its sparsification strength is determined by the threshold of entry removal.
- **Multi-layer bounds:** When applied across multiple GNN layers, UNIFEWS provides an effective approximation to the graph smoothing optimization. Its output representation is close to the original objective.
- **Improvements on efficiency and efficacy:** UNIFEWS offers inherent advantages in terms of GNN efficacy and efficiency, notably mitigating the over-smoothing issue and facilitating enhanced joint sparsity.

A. UNIFEWS for Graph Propagation

Key Intuition: Entries Denote Importance in Graph Computation. As depicted in Fig. 2(b), conventional graph sparsification methods for GNNs only provide fixed and uniform *graph-level* adjustments through the entire learning process. This lack of flexibility hinders their performance when employing to the recurrent update design with multiple GNN layers. Contrarily, *node-wise* models in Fig. 2(a) aim to personalize the graph-based propagation, but come with additional explicit calculations and impaired runtime efficiency. We are hence motivated to design a sparsification approach that (1) enables modifications in a fine-grained manner to further simplify the computation; and (2) can be seamlessly integrated into the matrix operation with negligible overhead.

For decoupled models, we mainly focus on the isolated graph propagation stage. We first express the propagation scheme Eq. (2) in an *entry-wise* manner on node u as:

$$\boldsymbol{p}_{(l+1)}[u] = \sum_{v\in\mathcal{N}(u)} \tau_{(l)}[u,v] = \sum_{v\in\mathcal{N}(u)} \boldsymbol{T}_{(l)}[u,v]\cdot\boldsymbol{p}_{(l)}[v], \quad (8)$$

where the propagation message $\tau_{(l)}[u,v]$ from node v to u is regarded as an entry. As illustrated in Fig. 1(a), in a single hop of GNN diffusion, edges carrying propagation messages exert varying impacts on neighboring nodes, whose importance is dependent on the graph topology. Messages with minor significance are usually considered redundant and can be eliminated to facilitate sparsity. From the perspective of matrix computation, this is achieved by omitting the particular message $\tau[u,v]$ in current aggregation based on an importance indicator such as its magnitude.

An example on the real-word graph cora is shown in Fig. 3(a), which demonstrates that the distribution of message values is correlated with the diffusion entries. Hence, message magnitude is effective in representing edge importance and determining entry removal. Setting entries in the diffusion matrix $\boldsymbol{T}$ to zero according to Eq. (9) implies removing the corresponding edges from the graph diffusion process. Consequently, messages with small magnitudes are prevented from propagating to neighboring nodes and composing the output embedding.

Edge Pruning for Single Layer. In order to perform pruning on entry $\tau[u,v]$, the diffusion matrix $\boldsymbol{T}$ can be utilized to record the pruning information, which is exactly the concept of graph sparsification. Given a universal magnitude threshold δ_a, zeroing out entries $|\tau[u,v]| < \delta_a$ is equivalent to sparsifying $\boldsymbol{T}$ with a node-wise threshold δ'_a as:

$$\hat{\boldsymbol{T}}[u,v] = \mathrm{thr}_{\delta'_a}\left(\boldsymbol{T}[u,v]\right)\cdot\boldsymbol{T}[u,v], \quad \delta'_a = \delta_a/|\boldsymbol{p}[v]|, \quad (9)$$

where the pruning function with an arbitrary threshold δ is $\mathrm{thr}_\delta(x) = 1$ if $|x| > \delta$, and $\mathrm{thr}_\delta(x) = 0$ otherwise. Sparsification by Eq. (9) typically has two concurrent effects: for graph computation, messages with small magnitudes are prevented from propagating to neighboring nodes and composing the output embedding; for graph topology, corresponding edges are removed from the graph diffusion process.

Next, we show that our edge pruning in Eq. (9) for one propagation layer can be considered as a spectral sparsification following Definition 2. In practice, the target edge sparsity $\eta_a \in [0,1]$ is usually determined by the realistic application. Let $\hat{\mathcal{E}}$ be the edge set achieved by UNIFEWS sparsification, there is $\eta_a = 1 - |\hat{\mathcal{E}}|/m$. The following theorem derives the required UNIFEWS threshold as well as the precision bound under a given sparsity:

*Theorem 2 (**Bound for UNIFEWS**):* Given a graph $\mathcal{G}$, embedding $\boldsymbol{p}$, and required edge sparsity η_a, the threshold δ_a can be decided by $\delta_a = C(1-\eta_a)^{-t}$, and the sparsified Laplacian $\hat{\boldsymbol{L}}$ is ϵ-similar to $\boldsymbol{L}$ with the approximation bound:

$$\epsilon = O\left(\eta_a(1-\eta_a)^{-t}\right), \tag{10}$$

where C and t are positive constants. C is determined by embedding values $\|\boldsymbol{p}\|$, and t depicts the degree distribution in graph.

The proof of Theorem 2 and specific derivations of C and t can be found in the tech report [61]. Theorem 2 not only bridges UNIFEWS pruning and Laplacian smoothing, but presents the specific bounds as well. Its implication is that, the adaptive sparsification by UNIFEWS for a single layer qualifies as a graph spectral sparsifier bounded by ϵ. The sparsification threshold as well as the approximation bound can be determined once given the sparsity η_a. A sparsity closer to 1 results in a larger threshold value, more pruned entries, as well as a loose error guarantee.

Accumulated Pruning across Layers. Thanks to the adaptive property of the entry-wise pruning scheme, we are able to further perform fine-grained and gradual sparsification across propagation layers. Intuitively, a message deemed of minor significance in the current propagation is unlikely to propagate further and influence more distant nodes in subsequent layers. UNIFEWS hence offers progressively increasing sparsity throughout GNN layers.

Algorithm 1: UNIFEWS on Decoupled Propagation

```
Input: Graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$, diffusion $\boldsymbol{T}_{(l)}$, attribute $\boldsymbol{X}$, propagation hop $L$, graph sparsification threshold $\delta_a$
Output: Approximate embedding $\hat{\boldsymbol{P}}_{(L)}$
1  $\hat{\boldsymbol{P}}_{(0)} \leftarrow \boldsymbol{X}$, $\hat{\mathcal{E}}_{(-1)} \leftarrow \mathcal{E}$
2  for $l = 0$ to $L-1$ do
3      $\hat{\boldsymbol{P}}_{(l+1)} \leftarrow \hat{\boldsymbol{P}}_{(l)}$
4      $\hat{\mathcal{E}}_{(l)} \leftarrow \hat{\mathcal{E}}_{(l-1)}$, $\hat{\boldsymbol{T}}_{(l)} \leftarrow \boldsymbol{T}_{(l)}$
5      for all $u \in \mathcal{V}$ do                              ▷ [matrix op]
6          $\hat{\mathcal{N}}_{(l)}(u) \leftarrow \{v | (u,v) \in \hat{\mathcal{E}}_{(l)}, v \neq u\}$
7          for all $v \in \hat{\mathcal{N}}_{(l)}(u)$ do
8              if $|\hat{\boldsymbol{T}}_{(l)}[u,v]| < \delta_a / \|\hat{\boldsymbol{P}}_{(l)}[v]\|$ then
9                  $\hat{\mathcal{E}}_{(l)} \leftarrow \hat{\mathcal{E}}_{(l)} \setminus \{(u,v)\}$, $\hat{\boldsymbol{T}}_{(l)}[u,v] \leftarrow 0$
10             else
11                 $\hat{\boldsymbol{P}}_{(l+1)}[u] \leftarrow \hat{\boldsymbol{P}}_{(l+1)}[v] + \hat{\boldsymbol{T}}_{(l)}[u,v] \cdot \hat{\boldsymbol{P}}_{(l)}[v]$
12 return $\hat{\boldsymbol{P}}_{(L)}$
```

As depicted in Fig. 2(c), UNIFEWS iteratively applies sparsification to each layer, and the pruned diffusion matrix $\hat{\boldsymbol{T}}_{(l)}$ is inherited to the next layer for further edge removal. Denote the edge set corresponding to $\hat{\boldsymbol{T}}_{(l)}$ as $\hat{\mathcal{E}}_{(l)}$, there is $\hat{\mathcal{E}}_{(l)} \subseteq \hat{\mathcal{E}}_{(l-1)} \subseteq \cdots \subseteq \hat{\mathcal{E}}_{(0)} \subseteq \mathcal{E}$, which indicates a diminishing number of operations for deeper layers. Consequently, Eq. (8) is modified as the following to represent entry-level diffusion on the sparsified graph:

$$\hat{\boldsymbol{p}}_{(l+1)}[u] = \sum_{v \in \mathcal{N}(u)} \hat{\boldsymbol{T}}_{(l)}[u,v] \cdot \boldsymbol{p}_{(l)}[v] = \sum_{v \in \hat{\mathcal{N}}_{(l)}(u)} \tau_{(l)}[u,v], \tag{11}$$

where the neighborhood composed by remaining connections is $\hat{\mathcal{N}}_{(l)}(u) = \{v | (u,v) \in \hat{\mathcal{E}}_{(l)}\}$.

We illustrate the application of UNIFEWS on decoupled graph propagation in Algorithm 1 as highlighted components. Note that the nested loops starting from Line 5 are identical to the canonical graph propagation conducted by sparse-dense matrix multiplication in common GNNs. Compared to the normal propagation Line 11, it refrains the value update and prunes the corresponding edge for insignificant entries in Line 9. Hence, UNIFEWS can be implemented into the GNN computation to enhance efficiency without additional matrix-wise overhead. For multi-dimensional feature matrix, the pruning function Eq. (9) is performed by replacing $|\boldsymbol{p}[v]|$ with the specific norm $\|\boldsymbol{P}[v]\|$ across feature dimensions.

From the theoretical perspective, the following proposition states that, pruning and accumulating the sparsified graph across multiple hops still guarantees the graph smoothing approximation as Lemma 1:

*Proposition 3 (**Progressive UNIFEWS for graph propagation**):* For an L-hop propagation in Algorithm 1, if the final sparsification satisfies ϵ-similarity, then the overall graph smoothing is an ϵ-approximation to the original problem.

Compared to the single-layer case in Theorem 2, the error bound of multi-hop propagation is composed of approximation errors of both the diffusion and embedding values. As detailed in the tech report [61], Proposition 3 ensures that the graph approximation is bounded as long as the sparsification parameters satisfy Theorem 2 for each hop.

An empirical evaluation of multi-hop UNIFEWS on cora is shown in Fig. 3(b), which validates that the approximation error is affected by the sparsification threshold δ_a as well as propagation hops L. Meanwhile, even under aggressive sparsification, the error is still adequately bounded to yield meaningful learning outcomes.

Impact on Over-smoothing. Additionally, we note that the "error" induced by UNIFEWS is not necessarily harmful. Examining Eq. (4), excessive propagation of common GNNs can lead to a decline in performance known as the over-smoothing issue [62], where the graph regularization is dominant and meaningful node identity is lost. Conversely, by eliminating a portion of graph diffusion, UNIFEWS effectively alleviates the over-smoothing issue for GNN with deeper layers. We showcase its effect by depicting the embedding difference to optimization convergence in Fig. 3(c). With an increased number of hops, the output embedding tends towards an over-smoothed state. However, a stronger sparsification prevents the rapid smoothing and hereby contributes to better graph learning performance.

B. UNIFEWS *for Iterative Update*

Compared to decoupled designs, graph propagation and feature transformation in iterative GNN models are tightly integrated in each layer. Traditionally, this poses a challenge to GNN sparsification methods depicted in Fig. 2(b), as specialized schemes are necessary for simultaneously modifying the two components without impairing the performance. On the contrary, our UNIFEWS approach takes advantage of this architecture for jointly sparsifying the model towards a win-win situation in graph learning.

The sparsification of iterative UNIFEWS for the entire message-passing process is presented in Algorithm 2, where the difference from the common scheme is also highlighted. Its graph propagation stage is identical to Algorithm 1, except that the embedding $\hat{\boldsymbol{P}}_{(l)}$ is initialized by the previous representation $\hat{\boldsymbol{H}}_{(l)}$. Meanwhile, sparsification for weights is relatively straight-forward compared to graph edges, since weight matrices are structured and their approximation is well-studied as network pruning [16], [63], [64]. Given the embedding of the current layer $\hat{\boldsymbol{P}}_{(l)}$, we similarly rewrite the iterative update of Eq. (1) as:

$$\boldsymbol{H}_{(l+1)}[:,i] = \sigma\Big(\sum_{j=1}^{f} \boldsymbol{\omega}_{(l)}[j,i]\Big) = \sigma\Big(\sum_{j=1}^{f} \boldsymbol{W}_{(l)}[j,i] \cdot \boldsymbol{P}_{(l)}[:,j]\Big), \tag{12}$$

where $\boldsymbol{H}_{(l+1)}[:,i]$ denotes the i-th column vector of all nodes in $\boldsymbol{H}_{(l+1)}$, and the weight entry $\boldsymbol{W}_{(l)}[j,i]$ symbolizes the neuron mapping the j-th embedding feature to the i-th representation feature. UNIFEWS sparsification on the weight matrix can thus be presented in the entry-wise manner following weight threshold δ_w:

$$\hat{\boldsymbol{W}}[j,i] = \mathrm{thr}_{\delta'_w}\big(\boldsymbol{W}[j,i]\big) \cdot \boldsymbol{W}[j,i], \quad \delta'_w = \delta_w / \|\boldsymbol{P}[:,j]\|. \tag{13}$$

Algorithm 2: UNIFEWS on Iterative GNN

Input: Graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$, diffusion $\boldsymbol{T}_{(l)}$, attribute $\boldsymbol{X}$, network layer L, graph sparsification threshold δ_a, weight sparsification threshold δ_w

Output: Approximate representation $\hat{\boldsymbol{H}}_{(L)}$

```
$\hat{\boldsymbol{H}}_{(0)} \leftarrow \boldsymbol{X},\ \hat{\mathcal{E}}_{(-1)} \leftarrow \mathcal{E}$
for $l = 0$ to $L-1$ do
    $\hat{\boldsymbol{H}}_{(l+1)} \leftarrow \boldsymbol{0},\ \hat{\boldsymbol{P}}_{(l)} \leftarrow \hat{\boldsymbol{H}}_{(l)}$
    Acquire sparsified $\hat{\boldsymbol{P}}_{(l)}, \hat{\mathcal{E}}_{(l)}, \hat{\boldsymbol{T}}_{(l)}$ as in Algorithm 1
    for $i \leftarrow 1$ to $f$ do                                   ▷ [matrix op]
        for $j \leftarrow 1$ to $f$ and $\hat{\boldsymbol{W}}_{(l)}[j,i] \neq 0$ do
            if $|\hat{\boldsymbol{W}}_{(l)}[j,i]| < \delta_w / \|\hat{\boldsymbol{P}}_{(l)}[:,j]\|$ then
                $\hat{\boldsymbol{W}}_{(l)}[j,i] \leftarrow 0$
            else
                $\hat{\boldsymbol{H}}_{(l+1)}[:,i] \leftarrow \hat{\boldsymbol{H}}_{(l+1)}[:,i] + \hat{\boldsymbol{W}}_{(l)}[j,i] \cdot \hat{\boldsymbol{P}}_{(l)}[j]$
    $\hat{\boldsymbol{H}}_{(l+1)} \leftarrow \sigma\big(\hat{\boldsymbol{H}}_{(l+1)}\big)$
return $\hat{\boldsymbol{H}}_{(L)}$
```

On top of Proposition 3 for graph propagations, we are able to show that the representation under layer-progressive UNIFEWS for iterative networks containing both graph and weight operations satisfies the following proposition:

*Proposition 4 (**Progressive** UNIFEWS **for iterative update**):* For an L-round iterative update under Algorithm 2 with sparsification, the approximation error on output representation $\|\hat{\boldsymbol{H}}_{(L)} - \boldsymbol{H}_{(L)}\|_F$ is bounded by $O(\epsilon\|\boldsymbol{H}_{(L)}\|_F + \delta_w)$.

The detailed interpretation of Proposition 4 is discussed in the tech report [61]. In brief, the approximation of iterative layer representation is jointly bounded by factors depicting graph and weight sparsification processes. Hence, the unified pruning produces an advantageous approximation of the learned representations across GNN layers, which completes our framework for characterizing the approximation bound of UNIFEWS for general GNN schemes by examining the graph smoothing optimization throughout model learning.

Complexity Analysis. As shown in Algorithms 1 and 2, entry-level operations can be naturally inserted into the computation *without* additional overhead. For graph propagation, when the graph sparsity is $\eta_a = q_a/m$, where q_a is the number of removed edges, the computation complexity for propagation in each layer is reduced from $O(m)$ to $O\big((1-\eta_a)m\big)$. In particular, for iterative models, this applies to both time and memory overhead. For weight pruning regarding matrices $\boldsymbol{P}, \boldsymbol{H} \in \mathbb{R}^{n \times f}$ and $\boldsymbol{W} \in \mathbb{R}^{f \times f}$, let the pruning ratio of weight matrix be η_w. The sparsification scheme at least reduces complexity to $O\big((1-\eta_w)nf^2\big)$. The favorable merit of joint graph and weight sparsification by UNIFEWS is that, both the propagation result $\boldsymbol{P}$ and the weight multiplication product $\boldsymbol{H}$ enjoy sparsity from the previous input alternatively. In fact, as proven by [37], the scale of reduced computational operation can be advanced to be quadratic to $(1-\eta_w)$.

Comparative Discussion. We highlight three advantages of our UNIFEWS approach in the context of GNN learning. Firstly, realistic graph propagation and feature transformation are conducted in an entry-centric fashion. Therefore, entry-level operations can be naturally inserted into the computation *on the fly* by modifying the matrix multiplication operator. Secondly, different from previous node- or edge-dependent strategies, our sparsification operates on the messages being passed during propagation, which is inherently informative to foster effective pruning while retaining model *expressivity*. Lastly, the *progressive* pruning across layers is capable of promoting greater graph sparsity, thereby alleviating intrinsic drawbacks of the GNN architecture including over-smoothing and neighborhood explosion. We empirically showcase the improvement on entry sparsity in Section V-D.

V. EXPERIMENTAL EVALUATION

We implement UNIFEWS for sparsifying various GNN architectures and evaluate its performance against strong competitors in the scope of GNN graph and network simplification. In Sections V-B and V-C, we respectively highlight the accuracy and efficiency improvement of our approach, while the effects of hyperparameters are discussed in Section V-D.

A. *Experiment Settings*

Dataset. We adopt 6 representative datasets in total, including 3 small-scale ones: cora, citeseer, and pubmed [1] as well as 3 large-scale ones: arxiv, products, papers100m [65]. Statistics of the datasets can be found in Table II, where we follow the common settings including the graph preparing and training set splitting pipeline in these benchmarks. In the table, we incorporate self-loop edges and count undirected edges twice to better reflect the propagation overhead.

Metrics. We uniformly utilize transductive node classification accuracy as the evaluation metric of model prediction. To better observe the number of operations in GNN learning, the efficiency is assessed by computation time and floating-point operations (FLOPs), and 1FLOPs $\approx$ 2MACs. Evaluation are conducted on a server with 32 Intel Xeon CPUs (2.4GHz), an Nvidia A30 (24GB memory) GPU, and 512GB RAM.

Backbone Models. Since UNIFEWS is adaptable to a range of GNN architectures, we select 4 classic GNNs as our raw model architectures in this paper subject to UNIFEWS and baseline sparsification, while more backbones are available in our code. *Iterative* backbone models include:

- GCN [1] is the representative message-passing GNN with a diffusion matrix $\boldsymbol{T} = \tilde{\boldsymbol{A}}$ across all layers.
- GAT [50] learns a variable diffusion $\boldsymbol{T}$ for each layer by multi-head attention. We set the number of heads to 8.

For *decoupled* designs, we implement the pre-propagation version [53] of the following models:

- SGC [51] corresponds to GCN in spectral smoothing, but computes the propagation $\boldsymbol{P} = \tilde{\boldsymbol{A}}^L\boldsymbol{X}$ separately in advance.
- APPNP [54] accumulates and propagates the embedding with a decay factor $\boldsymbol{P} = \sum_{l=0}^{L-1} \alpha(1-\alpha)^l \tilde{\boldsymbol{A}}^l \boldsymbol{X}$.

Baseline Methods. For *iterative* models, we consider state of the arts in graph and joint compression, which are methods with the capability to produced sparsified graphs with smaller edge sets for both GNN training and inference:

- GLT [18] proposes concurrently sparsifying GNN structure by gradient-based masking on both adjacency and weights.
- GEBT [19] gradually discovers the small model during training. Its implementation is limited to the GCN backbone.

TABLE II: Statistics of graph datasets. f and N_c are the numbers of input attributes and label classes, respectively. "Train" is the portion of training set w.r.t. labeled nodes.

Dataset	Nodes n	Edges m	f	N_c	Train
cora	$2,485$	$12,623$	1433	7	50%
citeseer	$3,327$	$9,228$	3703	6	50%
pubmed	$19,717$	$88,648$	500	3	50%
arxiv	$169,343$	$2,315,598$	128	40	54%
products	$2,400,608$	$123,718,024$	100	47	8%
papers100m	$111,059,956$	$3,228,124,712$	128	172	78%

- CGP [38] iteratively prunes and regrows graph connections while exploiting irregular compression on weights.
- DSpar [12] employs one-shot graph sparsification according to a degree-based metric, which implies an upper bound for pruning rate. It does not perform weight pruning.
- Random refers to the baseline that removes entries with uniform probability based on the specified sparsity.

For the *decoupled* scheme, we mainly evaluate the graph sparsification. There are two propagation personalization techniques, both only available for the SGC backbone:

- NDLS [29] determines the hop by calculating the distance to convergence, which produces a customized propagation.
- NIGCN [30] offers better scalability by performing degree-based estimation on the node-wise propagation depth.

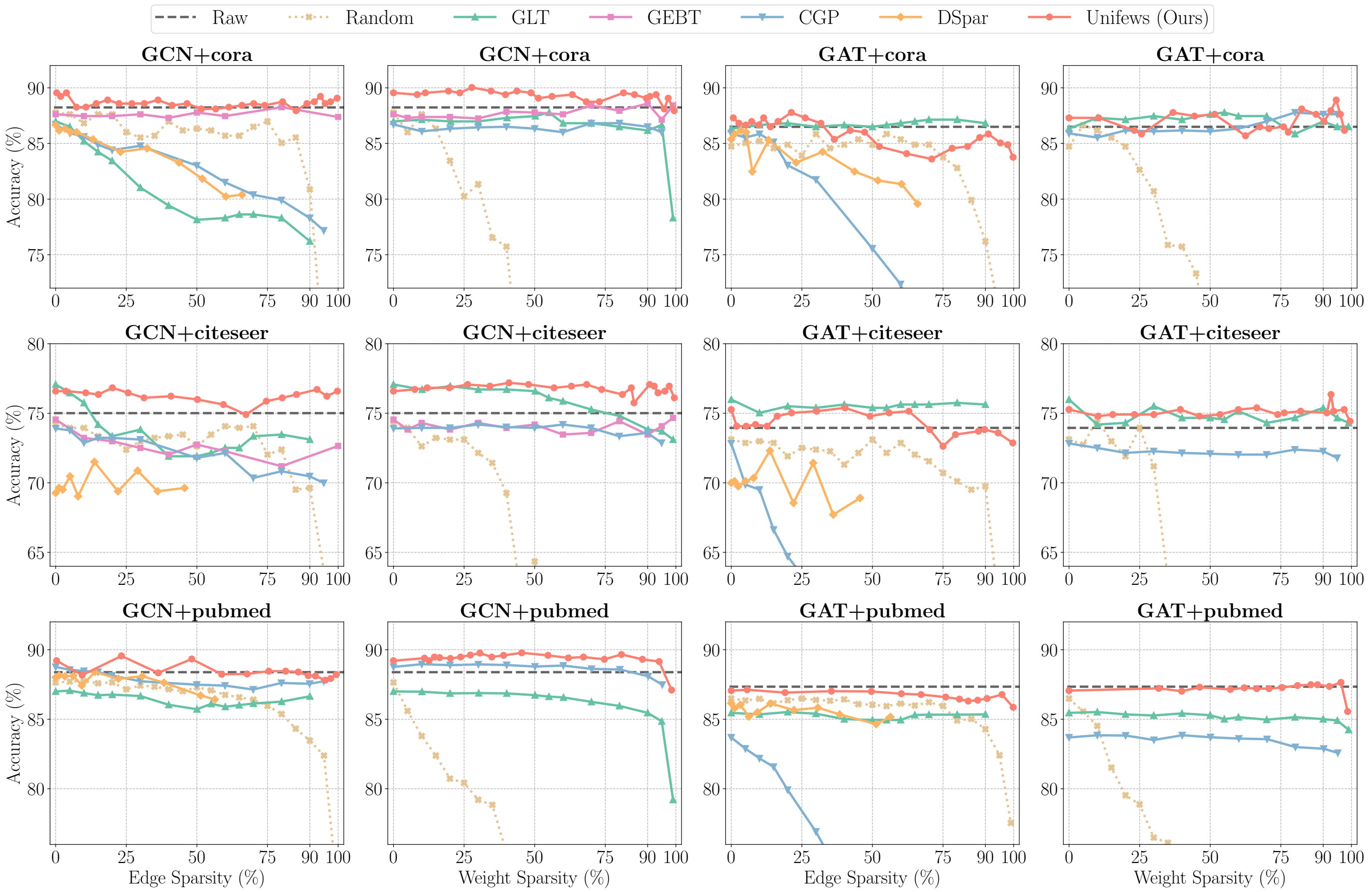
Hyperparameters. The common settings for dataset parameters, including graph normalization $r = 0.5$, model layer number $L = 2$, and layer width $f_{hidden} = 512$, are used across our experiments if not specified. For decoupled models, the number of propagation hops is 20. We employ full-batch and mini-batch training for iterative and decoupled methods, respectively. The total number of training epochs is uniformly 200 for all models, including pre-training or fine-tuning process in applicable methods. The batch size is 512 for small datasets and 16384 for large ones. We tune the edge and weight sparsity of evaluated models across their entire available ranges, and the pruning thresholds of UNIFEWS δ_a and δ_w are determined by Theorem 2.

B. *Performance Comparison*

Overview. We first separately apply only one part of sparsification, i.e., either edge or weight pruning, for more intuitive comparison. Fig. 4 presents accuracy results of iterative backbones and compression methods over representative datasets. GEBT encounters out of memory error on pubmed. For larger graphs, most of the baselines suffer from the out of memory error due to the expense of trainable adjacency matrix design and full-graph training process, hence the results are not presented. In contrast, we demonstrate the ability of UNIFEWS for boosting models on large graphs in Fig. 7 in later sections.

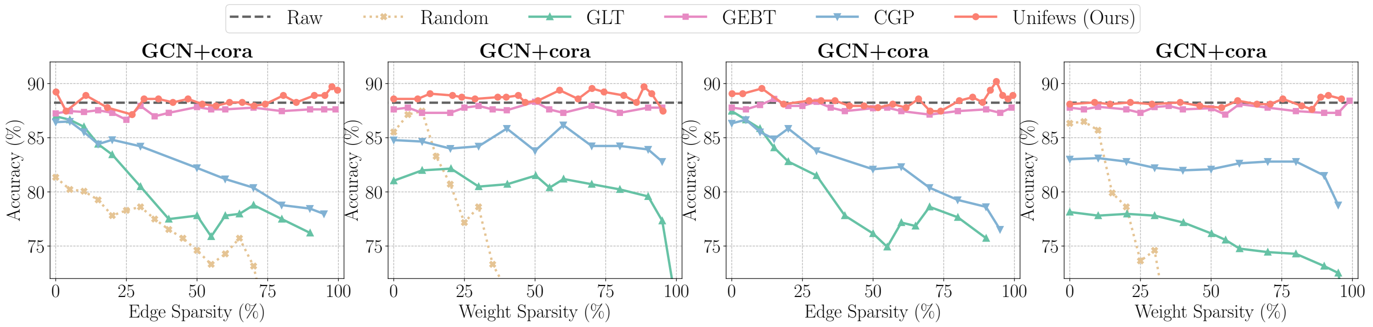
As an overview, for both graph and weight sparsification, our UNIFEWS approach successfully removes up to 90%-95% of entries while consistently attaining comparable or better prediction accuracy. Considering that the number of operations is proportional to density, it indicates that UNIFEWS is capable of achieving 10-20× computational reduction for GNN training and inference.

Edge Sparsification. In specific, with respect to edge sparsification on the *iterative* architecture in Figs. 4(a) and 4(c), UNIFEWS outperforms state-of-the-art graph and joint compression approaches in most backbone-dataset combinations. Typically, for relatively small ratios $\eta_a < 80\%$, models with UNIFEWS pruning achieve comparable or exceeding accuracy, aligning with our approximation analysis. For higher sparsity, UNIFEWS benefits from the skip connection design, which carries essential information of node identity and therefore retains accuracy no worse than the trivial transformation.



(a) GCN Edge Pruning (b) GCN Weight Pruning (c) GAT Edge Pruning (d) GAT Weight Pruning

Fig. 4: Accuracy results of UNIFEWS and baseline methods applied on iterative backbone models GCN and GAT over three small datasets. Columns of “Edge” and “Weight Sparsity” present the average results of models with *solely* edge and weight sparsification, respectively. Black dashed lines are the performance of backbone models with full graph and weights.

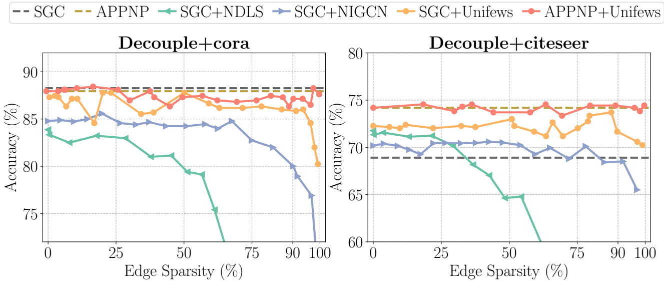


(a) Edge Pruning, $\eta_w = 30\%$ (b) Weight Pruning, $\eta_a = 30\%$ (c) Edge Pruning, $\eta_w = 50\%$ (d) Weight Pruning, $\eta_a = 50\%$

Fig. 5: Accuracy of *joint sparsification* of models with varying edge and weight sparsification, while fixing the other sparsity. Black dashed lines are the performance of backbone GCN.

On the contrary, most of the competitors experience significant accuracy drop on one or more datasets. On cora and citeseer where graph topology is important, aggressive pruning causes noticeable performance loss. CGP and DSpar exhibit poor utility on the GAT backbone since its variable connections are more vulnerable to removal. Additionally, the comparison with random sparsification indicates that, the entry-wise scheme is particularly effective for small ratios, as the randomized pruning even surpasses dedicated methods for some circumstances. For high edge sparsity, the UNIFEWS preservation of dominant edges and identity mapping is critical to accuracy, which validates the advantage of our design.

Similarly, for edge sparsification on *decoupled* propagation in Fig. 6, UNIFEWS is able to preserve remarkable accuracy even with high sparsity, exhibiting its superiority over personalized propagation methods, especially for high sparsity.



(a) Edge Pruning on cora (b) Edge Pruning on citeseer

Fig. 6: Accuracy of graph sparsification on *decoupled* models over cora and citeseer. Unifews is employed with solely edge removal. Black and brown dashed lines are the performance of backbone SGC and APPNP.

On citeseer, it also raises SGC accuracy by 2% through mitigating the over-smoothing issue. We hence conclude that, compared to heuristic pruning schemes, Unifews successfully identifies and removes unimportant graph edges without sacrificing eventual learning efficacy, thanks to its fine-grained and adaptive scheme embedded in graph computation.

Weight Sparsification. Figs. 4(b) and 4(d) display the efficacy of network pruning on iterative GNNs. For all combinations of models and graphs, Unifews achieves top-tier accuracy with a wide range of weight sparsity. For small ratios, the model produces up to 3% accuracy gain over the bare GNN, resembling the benefit of neural network compression [66]. Most evaluated baselines also maintain low error rate compared to their performance in graph pruning, indicating that the GNN weights are relatively redundant and are suitable for substantial compression. Thanks to the redundancy in GNN architecture, baselines including GEBT and CGP present strong performance in certain cases compared to edge sparsification. Random removal fails for high weight sparsity, which implies that the magnitude-based scheme of Unifews is the key to maintain model effectiveness.

Joint Sparsification. We then showcase the ability of Unifews in conducting unified pruning on graph edges and network weights concurrently. Accuracy comparison is provided by Fig. 5 with applicable methods for joint compression. It is noticeable that Unifews retains comparable or better accuracy than the backbone GCN with up to 3% improvement. Most baseline methods only obtain suboptimal accuracy especially for weight pruning, which is affected by their comparatively poor graph sparsification. While GEBT mostly retains effectiveness on cora, it is highly limited by the specific architecture and high training overhead. The evaluation further highlights the advantage of Unifews in considering the two GNN operation stages in a unified manner.

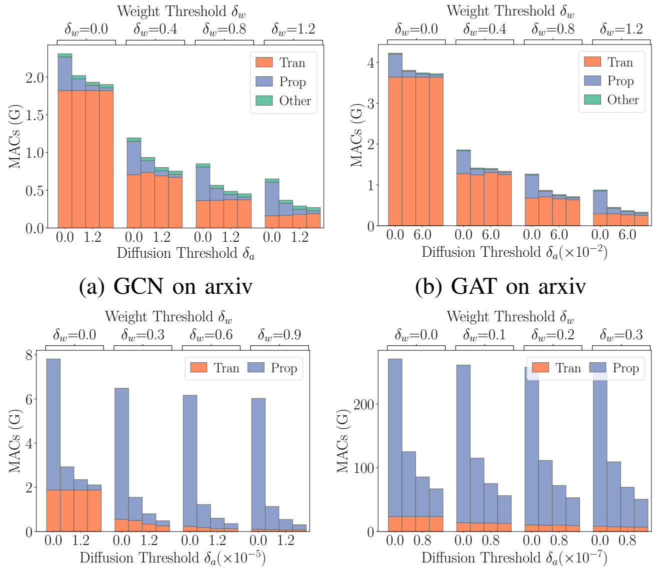
C. Efficiency Analysis

FLOPs Comparison. As graph computation is the primary bottleneck for large-scale tasks, we particularly study the efficiency improvement utilizing decoupled models in Table III. We utilize a representative graph sparsification ratio $\eta_a = 50\%$, and compare the running time as well as FLOPs for decoupled models. Evaluation on the propagation FLOPs implies that, Unifews is effective in producing operational reduction proportional to the sparsity, i.e., saving 50% computation FLOPs. It also achieves higher compression such as for APPNP over products, by recognizing more unnecessary edges than required.

Contrarily, NDLS suffers from out of memory error on large datasets due to its complex calculation and implementation, while NIGCN does not guarantee decreased computation because of its expansionary propagation design. Table III also presents the transformation FLOPs for reference, where the sparsified embedding of Unifews also benefits the downstream computation. We nonetheless remark that, although the

TABLE III: Evaluation results of $\eta_a = 50\%$ graph sparsification on decoupled models. Unifews is applicable to both SGC and APPNP backbones, while two baseline methods are only available on the former one. "Acc" and "Time" are the prediction accuracy (%) and propagation time (in seconds), respectively. "FLOPs" separately presents the operational complexity (in GMACs) for propagation and transformation on all nodes. "OOM" stands for out of memory error. "Improvement" is the comparison between Unifews and the corresponding backbone model, where improved accuracy is marked in **bold** fonts.

Dataset	**cora**			**citeseer**			**pubmed**			**arxiv**			**products**			**papers100m**		
Nodes n	$2,485$			$3,327$			$19,717$			$169,343$			$2,400,608$			$111,059,956$		
Edges m	$12,623$			$9,228$			$88,648$			$2,315,598$			$123,718,024$			$3,228,124,712$		
Metric	Acc	Time	FLOPs	Acc	Time	FLOPs	Acc	Time	FLOPs	Acc	Time	FLOPs	Acc	Time	FLOPs	Acc	Time	FLOPs
SGC	85.8	0.13	0.36+2.2	67.7	0.44	0.68+2.8	83.0	0.36	0.89+2.3	68.8	3.9	5.9+15.0	79.1	289.6	247.4+434.4	63.3	19212	17.7+253.6
+NDLS	80.3	1362	0.19+2.7	64.7	1940	0.33+4.3	77.0	4717	0.42+2.0	(OOM)			(OOM)			(OOM)		
+NIGCN	84.2	0.45	0.22+3.0	70.4	0.47	0.28+2.1	85.0	9.2	0.44+2.0	63.7	87.6	15.6+14.7	77.9	1026	137.2+182.0	53.7	1770	110.7+238.6
+Unifews	**86.0**	0.10	0.18+1.2	**73.0**	0.26	0.35+1.7	83.0	0.24	0.47+2.0	**69.4**	1.5	3.0+13.3	78.5	203.1	124.0+186.9	63.1	192.4	5.3+143.8
Improvement	0.2	1.3×	2.0×, 1.9×	5.3	1.7×	2.0×, 1.7×	0.0	1.5×	1.9×, 1.2×	0.6	2.6×	2.0×, 1.1×	-0.5	1.4×	2.0×, 2.3×	-0.2	99.8×	3.3×, 1.8×
APPNP	86.2	0.15	0.36+2.2	71.6	0.43	0.68+2.8	87.6	0.33	0.89+2.3	64.8	2.6	5.9+20.9	72.5	248.5	247.4+269.4	60.9	15305	17.7+247.7
+Unifews	**86.5**	0.08	0.18+1.8	**73.7**	0.21	0.31+2.3	**88.0**	0.26	0.43+1.8	**65.0**	0.93	3.0+15.0	**76.9**	58.5	31.7+186.9	**62.8**	178.5	8.9+241.8
Improvement	0.4	1.8×	2.0×, 1.2×	2.1	2.0×	2.2×, 1.2×	0.4	1.3×	2.1×, 1.3×	0.2	2.8×	1.9×, 1.4×	4.5	4.2×	7.8×, 1.4×	1.9	85.7×	2.0×, 1.0×



(a) GCN on arxiv

(b) GAT on arxiv

(c) SGC on arxiv

(d) SGC on products

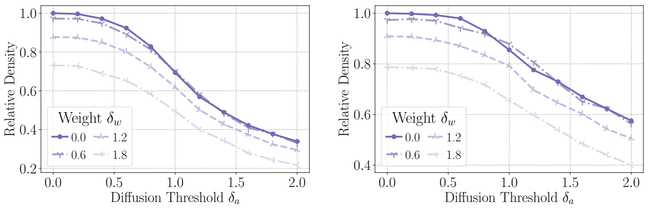
Fig. 7: Composition of full-graph inference FLOPs with respect to propagation and transformation operations under varying UNIFEWS edge and weight thresholds. Iterative models: **(a)** GCN and **(b)** GAT over arxiv. Decoupled models: SGC over **(c)** arxiv and **(d)** products datasets. For better presentation, model width is set to 64 in this experiment.

FLOPs value for transformation stage appears to be on the same level with propagation, in practice it can be efficiently processed utilizing parallel computation on devices such as GPU [67], [68]. Hence the GNN scalability bottleneck, especially on large datasets, is still the graph propagation. In this context, the ability of UNIFEWS in reducing propagation operations is of unique importance.

Runtime. Within each method in Table III, the propagation time is correlated with its FLOPs, and escalates rapidly with the data size due to the nature of graph operations. Comparison across methods demonstrates that UNIFEWS excels in efficiently conducting graph propagation with regard to both FLOPs and execution time. It realizes significant acceleration on large graphs by favorably reducing the graph scale across all layers, which benefits the system workload such as better entry-level access. The papers100m result highlights the superiority of UNIFEWS with $85-100\times$ improvement over the backbone and $9\times$ speed-up over NIGCN.

FLOPs Breakdown. To specifically evaluate the efficiency enhancement, in Fig. 7, we separately assess the FLOPs related to graph propagation and network transformation for different backbone models and datasets under UNIFEWS sparsification. For *iterative models*, Figs. 7(a) and 7(b) imply that the majority of computational overhead is network transformation, even on graphs as large as arxiv. Consequently, weight compression is essential for remarkably reducing GNN operations.

On the other hand, graph propagation becomes the bottleneck in *decoupled designs*, and is increasingly significant on graphs with greater scales. This is because of the larger number of propagation hops $L = 20$ for these structures. In this case, UNIFEWS is effective in saving computational cost by simplifying the propagation and bypassing unnecessary operations, with over $20\times$ and $5\times$ joint reduction on arxiv and products, respectively. We also discover the benefit of joint pruning that a higher threshold results in smaller FLOPs of both operations in Figs. 7(c) and 7(d), which signifies the win-win situation brought by increased sparsity. By combining these two sparsifications, we summarize that the unified scheme of UNIFEWS is capable of mitigating the computational overhead for both propagation- and transformation-heavy tasks.



(a) Message $\boldsymbol{\tau}_{(0)}$

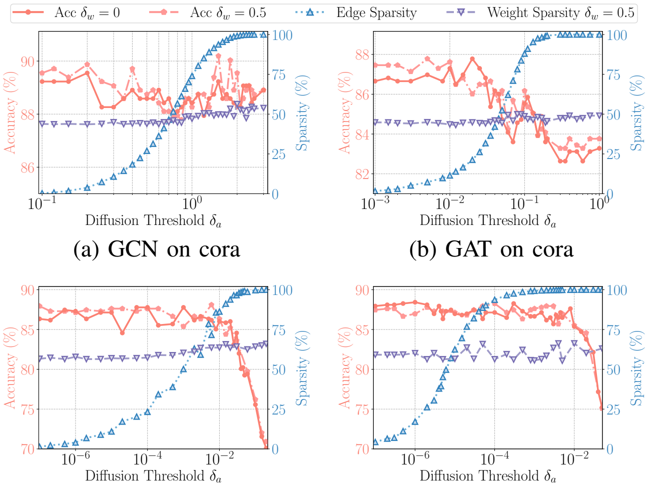
(b) Representation $\boldsymbol{H}_{(1)}$

Fig. 8: Density of intermediate entries during joint edge and weight sparsification at varying threshold values, evaluated by the ratio of matrix norm to the unpruned one on a 2-layer GCN over cora. **(a)** Density of edge-wise message $\|\hat{\boldsymbol{\tau}}_{(0)}\|/\|\boldsymbol{\tau}_{(0)}\|$ relative to the raw one in the first layer $l = 0$. **(b)** Density of node-wise representation $\|\hat{\boldsymbol{H}}_{(1)}\|_F/\|\boldsymbol{H}_{(1)}\|_F$ relative to the raw one output by the first layer.

D. *Effect of Hyperparameters*

Sparsity. We then present in-depth exploration regarding the effect of parameters in UNIFEWS for elaborating its performance in terms of efficiency and efficacy improvements, especially highlighting the superiority of joint sparsification. Fig. 8 investigates the entry-wise sparsity of specific intermediate results in the process of GNN computation, displaying the relative density of the edge message and node representation matrices, respectively. It can be clearly observed that the entry sparsity is enhanced with the increase of both two pruning ratios, signifying our theoretical analysis that UNIFEWS not only directly shrinks edges and weights, but promotes the sparsity of the product matrix as well. It also supports our claim that UNIFEWS is superior in employing dual sparsification where the two stages benefit each other alternatively and enjoy a win-win situation brought by the increased sparsity.

Sparsification Thresholds δ_a and δ_w. The thresholds are pivotal to UNIFEWS compression, affecting both efficacy and efficiency of the produced model. Fig. 9 presents the changes of inference accuracy and dual sparsity of GNN models under graph and joint sparsification with varying adjacency thresholds δ_a. Accuracy in the plot follows the conclusion in previous evaluation, that it only degrades above extreme sparsity $\eta_a > 95\%$. The experiment reveals that, the relation between edge sparsity and adjacency threshold aligns with Theorem 2, and decoupled models typically require a larger range of threshold to traverse the sparsity range, which is be-



(a) GCN on cora

(b) GAT on cora

(c) SGC on cora

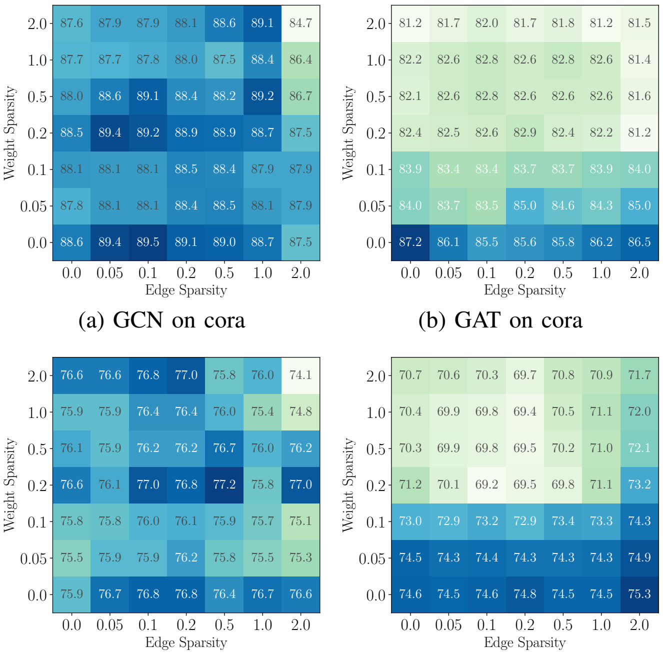
(d) APPNP on cora

Fig. 9: Sensitivity of joint sparsification thresholds on four models over cora. We respectively set the weight ratio δ_w to 0 and 0.5, then evaluate the accuracy, edge sparsity, and weight sparsity of the model. Note that the x-axis representing diffusion threshold is on a logarithmic scale.

cause of the wider distribution of their entry values throughout the deeper propagation. Interestingly, the weight sparsity when $\delta_w = 0.5$ also increases under high edge pruning ratios. The pattern is also observed in the reverse case. We deduce that the reciprocal enhancement in graph and model sparsity is brought by the unified sparsification of UNIFEWS, which conforms to our analysis as well as previous studies [37].

For effectiveness, the impact of jointly changing δ_a and δ_w is displayed in Fig. 10. Intuitively, GCN is more resilient to UNIFEWS sparsification, considering its relatively high redundancy of the wide distribution of entry values. In comparison, GAT is more sensitive to weight removal, that beyond a certain threshold δ_w, its accuracy drops significantly. This observation suggests that the value of learnable attention weights in GAT are highly concentrated, and selecting an appropriate sparsity is critical to its performance.

Network and Propagation Layer L. Since UNIFEWS adopts layer-dependent graphs with incremental sparsity for GNN propagation, we additionally evaluate its performance on models with varying layers. Fig. 11(a) is the result of iteratively applying UNIFEWS edge sparsification to SGC with different propagation depths. The backbone model without pruning $\delta_a = 0$ typically suffers from the over-smoothing issue under large hop numbers. On the contrary, as elaborated in Section IV-A, UNIFEWS is powerful for identifying and eliminating unimportant propagations, especially for larger hops, and thereby prevents information loss. With respect to architectural compression Fig. 11(b), it is noticeable that UNIFEWS promotes model performance and average weight sparsity at the same time for deeper layers, effectively reducing network redundancy.



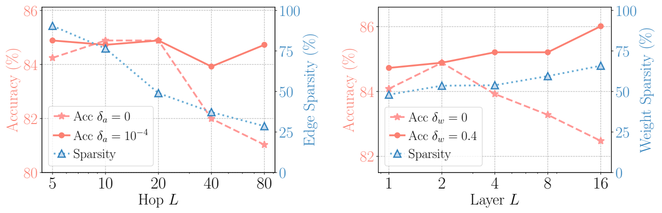
(a) GCN on cora

(b) GAT on cora

(c) GCN on citeseer

(d) GAT on citeseer

Fig. 10: Accuracy of varying joint sparsification thresholds on GCN and GAT over cora and citeseer.



(a) Propagation Hop

(b) Network Layer

Fig. 11: Sensitivity of propagation and network transformation layers evaluated on SGC over cora. **(a)** Impact on accuracy and average edge sparsity of varying numbers of propagation hops. **(b)** Impact on accuracy and average weight sparsity of varying numbers of network layers. Note that the x-axis representing layers is on a logarithmic scale.

VI. CONCLUSION

In this work, we present UNIFEWS, an entry-wise GNN sparsification with a unified framework for graph edges and model weights. UNIFEWS adaptively simplifies both graph propagation and feature transformation throughout the GNN pipeline and remarkably improves efficiency with little expense. In theory, we showcase that the layer-dependent UNIFEWS provides an effective estimation on the graph learning process with a close optimization objective, which is favorable for multi-layer GNN updates. Comprehensive experiments underscore the superiority of UNIFEWS in terms of efficacy and efficiency, including comparable or improved accuracy, $90-95\%$ operational reduction, and up to 100× faster computation.

References

[1] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” in *5th International Conference on Learning Representations*, 2017, pp. 1–14.

[2] W. L. Hamilton, R. Ying, and J. Leskovec, “Inductive representation learning in large attributed graphs,” in *30th Advances in Neural Information Processing Systems*, Long Beach, CA, USA, October 2017, pp. 1–11.

[3] P. Veličković, A. Casanova, P. Liò, G. Cucurull, A. Romero, and Y. Bengio, “Graph attention networks,” in *6th International Conference on Learning Representations*, 2018, pp. 1–12.

[4] M. Zhang and Y. Chen, “Link prediction based on graph neural networks,” in *31st Advances in Neural Information Processing Systems*, 2018, pp. 5165–5175.

[5] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, “A comprehensive survey on graph neural networks,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, January 2021.

[6] M. Chen, Z. Wei, B. Ding, Y. Li, Y. Yuan, X. Du, and J. R. Wen, “Scalable graph neural networks via bidirectional propagation,” in *33rd Advances in Neural Information Processing Systems*, vol. 2020-Decem, 2020, pp. 1–14.

[7] X. Liu, M. Yan, L. Deng, G. Li, X. Ye, D. Fan, S. Pan, and Y. Xie, “Survey on graph neural network acceleration: An algorithmic perspective,” in *Proceedings of the 31st International Joint Conference on Artificial Intelligence*, April 2022.

[8] S. Zhang, A. Sohrabizadeh, C. Wan, Z. Huang, Z. Hu, Y. Wang, Yingyan, Lin, J. Cong, and Y. Sun, “A survey on graph neural network acceleration: Algorithms, systems, and customized hardware,” June 2023.

[9] Y. Liu, T. Safavi, A. Dighe, and D. Koutra, “Graph summarization methods and applications: A survey,” *ACM Computing Surveys*, vol. 51, no. 3, pp. 1–34, May 2019.

[10] C. Zheng, B. Zong, W. Cheng, D. Song, J. Ni, W. Yu, H. Chen, and W. Wang, “Robust graph representation learning via neural sparsification,” in *37th International Conference on Machine Learning*, vol. 119. PMLR, 2020, pp. 11 458–11 468.

[11] H. Zeng, M. Zhang, Y. Xia, A. Srivastava, A. Malevich, R. Kannan, V. Prasanna, L. Jin, and R. Chen, “Decoupling the depth and scope of graph neural networks,” in *34th Advances in Neural Information Processing Systems*, vol. 34. Curran Associates, Inc., 2021, pp. 19 665–19 679.

[12] Z. Liu, K. Zhou, Z. Jiang, L. Li, R. Chen, S.-H. Choi, and X. Hu, “Dspar : An embarrassingly simple strategy for efficient gnn training and inference via degree-based sparsification,” *Transactions on Machine Learning Research*, July 2023.

[13] D. Li, T. Yang, L. Du, Z. He, and L. Jiang, “Adaptivegcn: Efficient gcn through adaptively sparsifying graphs,” in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. Virtual Event Queensland Australia: ACM, October 2021, pp. 3206–3210.

[14] J. Li, T. Zhang, H. Tian, S. Jin, M. Fardad, and R. Zafarani, “Graph sparsification with graph convolutional networks,” *International Journal of Data Science and Analytics*, vol. 13, no. 1, pp. 33–46, January 2022.

[15] W. Jin, L. Zhao, S. Zhang, Y. Liu, J. Tang, and N. Shah, “Graph condensation for graph neural networks,” in *10th International Conference on Learning Representations*, September 2022.

[16] B. L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, “Model compression and hardware acceleration for neural networks: A comprehensive survey,” *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, April 2020.

[17] H. Zhou, A. Srivastava, H. Zeng, R. Kannan, and V. Prasanna, “Accelerating large scale real-time gnn inference using channel pruning,” *Proceedings of the VLDB Endowment*, vol. 14, no. 9, pp. 1597–1605, 2021.

[18] T. Chen, Y. Sui, X. Chen, A. Zhang, and Z. Wang, “A unified lottery ticket hypothesis for graph neural networks,” in *38th International Conference on Machine Learning*, vol. 139. PMLR, 2021, pp. 1695–1706.

[19] H. You, Z. Lu, Z. Zhou, Y. Fu, and Y. Lin, “Early-bird gcns: Graph-network co-optimization towards more efficient gcn training and inference via drawing early-bird lottery tickets,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 8, pp. 8910–8918, June 2022.

[20] K. Wang, Y. Liang, P. Wang, X. Wang, P. Gu, J. Fang, and Y. Wang, “Searching lottery tickets in graph neural networks: A dual perspective,” in *11th International Conference on Learning Representations*, 2023.

[21] J. Chen, T. Ma, and C. Xiao, “Fastgcn: Fast learning with graph convolutional networks via importance sampling,” in *6th International Conference on Learning Representations*, 2018, pp. 1–15.

[22] D. Zou, Z. Hu, Y. Wang, S. Jiang, Y. Sun, and Q. Gu, “Layer-dependent importance sampling for training deep and large graph convolutional networks,” in *33rd Advances in Neural Information Processing Systems*, 2019.

[23] R. S. Srinivasa, C. Xiao, L. Glass, J. Romberg, and J. Sun, “Fast graph attention networks using effective resistance based graph sparsification,” October 2020.

[24] E. Kosman, J. Oren, and D. Di Castro, “Lsp: Acceleration of graph neural networks via locality sensitive pruning of graphs,” in *2022 IEEE International Conference on Data Mining Workshops (ICDMW)*. Orlando, FL, USA: IEEE, November 2022, pp. 690–697.

[25] D. A. Spielman and N. Srivastava, “Graph sparsification by effective resistances,” *SIAM Journal on Computing*, vol. 40, no. 6, pp. 1913–1926, January 2011.

[26] C. Deng, Z. Zhao, Y. Wang, Z. Zhang, and Z. Feng, “Graphzoom: A multi-level spectral approach for accurate and scalable graph embedding,” in *8th International Conference on Learning Representations*, February 2020.

[27] C. Cai, D. Wang, and Y. Wang, “Graph coarsening with neural networks,” in *9th International Conference on Learning Representations*, February 2021.

[28] Z. Huang, S. Zhang, C. Xi, T. Liu, and M. Zhou, “Scaling up graph neural networks via graph coarsening,” in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, vol. 1. Virtual Event Singapore: ACM, August 2021, pp. 675–684.

[29] W. Zhang, M. Yang, Z. Sheng, Y. Li, W. Ouyang, Y. Tao, Z. Yang, and B. Cui, “Node dependent local smoothing for scalable graph learning,” in *34th Advances in Neural Information Processing Systems*, 2021, pp. 20 321–20 332.

[30] K. Huang, J. Tang, J. Liu, R. Yang, and X. Xiao, “Node-wise diffusion for scalable graph learning,” in *Proceedings of the ACM Web Conference 2023*. Austin TX USA: ACM, April 2023, pp. 1723–1733.

[31] N. Liao, D. Mo, S. Luo, X. Li, and P. Yin, “Scara: Scalable graph neural networks with feature-oriented optimization,” *Proceedings of the VLDB Endowment*, vol. 15, no. 11, pp. 3240–3248, 2022.

[32] N. Liao, D. Mo, S. Luo, X. Li, and P. Yin, “Scalable decoupling graph neural network with feature-oriented optimization,” *The VLDB Journal*, December 2023.

[33] K.-H. Lai, D. Zha, K. Zhou, and X. Hu, “Policy-gnn: Aggregation optimization for graph neural networks,” in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. Virtual Event CA USA: ACM, August 2020, pp. 461–471.

[34] I. Spinelli, S. Scardapane, and A. Uncini, “Adaptive propagation graph convolutional network,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 10, pp. 4755–4760, October 2021.

[35] X. Miao, W. Zhang, Y. Shao, B. Cui, L. Chen, C. Zhang, and J. Jiang, “Lasagne: A multi-layer graph convolutional network framework via node-aware deep architecture,” *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–1, 2021.

[36] W. Zhang, Z. Yin, Z. Sheng, Y. Li, W. Ouyang, X. Li, Y. Tao, Z. Yang, and B. Cui, “Graph attention multi-layer perceptron,” in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. Washington DC USA: ACM, August 2022, pp. 4560–4570.

[37] S. Zhang, M. Wang, P.-Y. Chen, S. Liu, S. Lu, and M. Liu, “Joint edge-model sparse learning is provably efficient for graph neural networks,” in *11th International Conference on Learning Representations*, 2023.

[38] C. Liu, X. Ma, Y. Zhan, L. Ding, D. Tao, B. Du, W. Hu, and D. P. Mandic, “Comprehensive graph gradual pruning for sparse training in graph neural networks,” *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–15, 2023.

[39] J. Frankle and M. Carbin, “The lottery ticket hypothesis: Finding sparse, trainable neural networks,” in *7th International Conference on Learning Representations*, 2019, pp. 1–42.

[40] B. Hui, D. Yan, X. Ma, and W.-S. Ku, “Rethinking graph lottery tickets: Graph sparsity matters,” in *11th International Conference on Learning Representations*, 2023.

[41] Y. Sui, X. Wang, T. Chen, X. He, and T.-S. Chua, "Inductive lottery ticket learning for graph neural networks," October 2021.

[42] M. Ding, K. Kong, J. Li, C. Zhu, J. P. Dickerson, F. Huang, and T. Goldstein, "Vq-gnn: A universal framework to scale up graph neural networks using vector quantization," in *34th Advances in Neural Information Processing Systems*, 2021.

[43] M. Bahri, G. Bahl, and S. Zafeiriou, "Binary graph neural networks," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 9492–9501.

[44] J. Wang, Y. Wang, Z. Yang, L. Yang, and Y. Guo, "Bi-gcn: Binary graph convolutional network," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 1561–1570.

[45] W. Huang, T. Zhang, Y. Rong, and J. Huang, "Adaptive sampling towards fast graph representation learning," in *31st Advances in Neural Information Processing Systems*, vol. 31. Curran Associates, Inc., 2018.

[46] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "Graphsaint: Graph sampling based inductive learning method," in *8th International Conference on Learning Representations*, July 2020.

[47] Y. Rong, W. Huang, T. Xu, and J. Huang, "Dropedge: Towards deep graph convolutional networks on node classification," in *8th International Conference on Learning Representations*, March 2020.

[48] T. Fang, Z. Xiao, C. Wang, J. Xu, X. Yang, and Y. Yang, "Dropmessage: Unifying random dropping for graph neural networks," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 4, pp. 4267–4275, June 2023.

[49] H. Ding, Z. Wei, and Y. Ye, "Large-scale spectral graph neural networks via laplacian sparsification," 2024.

[50] S. Brody, U. Alon, and E. Yahav, "How attentive are graph attention networks?" in *10th International Conference on Learning Representations*, January 2022.

[51] F. Wu, T. Zhang, A. H. de Souza, C. Fifty, T. Yu, and K. Q. Weinberger, "Simplifying graph convolutional networks," in *36th International Conference on Machine Learning*, vol. 2019-June. PMLR, 2019, pp. 11 884–11 894.

[52] J. Gasteiger, S. Weißenberger, and S. Günnemann, "Diffusion improves graph learning," in *32nd Advances in Neural Information Processing Systems*, 2019.

[53] H. Wang, M. He, Z. Wei, S. Wang, Y. Yuan, X. Du, and J.-R. Wen, "Approximate graph propagation," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. Virtual Event Singapore: ACM, August 2021, pp. 1686–1696.

[54] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," in *7th International Conference on Learning Representations*, 2019, pp. 1–15.

[55] Y. Ma, X. Liu, T. Zhao, Y. Liu, J. Tang, and N. Shah, "A unified view on graph neural networks as graph signal denoising," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. Virtual Event Queensland Australia: ACM, October 2021, pp. 1202–1211.

[56] M. Zhu, X. Wang, C. Shi, H. Ji, and P. Cui, "Interpreting and unifying graph neural networks with an optimization framework," in *Proceedings of the Web Conference 2021*. Ljubljana Slovenia: ACM, April 2021, pp. 1215–1226.

[57] J. Batson, D. A. Spielman, N. Srivastava, and S.-H. Teng, "Spectral sparsification of graphs: theory and algorithms," *Communications of the ACM*, vol. 56, no. 8, pp. 87–94, August 2013.

[58] V. Sadhanala, Y.-X. Wang, and R. J. Tibshirani, "Graph sparsification approaches for laplacian smoothing," in *19th International Conference on Artificial Intelligence and Statistics*. Cadiz, Spain: PMLR, May 2016, pp. 1250–1259.

[59] D. Calandriello, I. Koutis, A. Lazaric, and M. Valko, "Improved large-scale graph learning through ridge spectral sparsification," in *35th International Conference on Machine Learning*, vol. 80. Stockholm, Sweden: PMLR, 2018.

[60] N. Charalambides and A. O. Hero, "Graph sparsification by approximate matrix multiplication," in *2023 IEEE Statistical Signal Processing Workshop (SSP)*. Hanoi, Vietnam: IEEE, July 2023, pp. 180–184.

[61] N. Liao, Z. Yu, R. Zeng, and S. Luo, "Unifews technical report," 2024. [Online]. Available: https://github.com/nyLiao/Unifews/blob/main/tech_report.pdf

[62] Q. Li, Z. Han, and X. M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[63] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *28th Advances in Neural Information Processing Systems*, June 2015, pp. 1135–1143.

[64] S. Srinivas and R. V. Babu, "Data-free parameter pruning for deep neural networks," in *Procedings of the British Machine Vision Conference 2015*. Swansea: British Machine Vision Association, 2015, pp. 31.1–31.12.

[65] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, J. Leskovec, R. Barzilay, P. Battaglia, Y. Bengio, M. Bronstein, S. Günnemann, W. Hamilton, T. Jaakkola, S. Jegelka, M. Nickel, C. Re, L. Song, J. Tang, M. Welling, and R. Zemel, "Open graph benchmark: Datasets for machine learning on graphs," in *33rd Advances in Neural Information Processing Systems*, vol. 2020-Decem, 2020, pp. 1–34.

[66] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste, "Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks," January 2021.

[67] H. Liu, S. Lu, X. Chen, and B. He, "G3: When graph neural networks meet parallel graph processing systems on gpus," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 2813–2816, August 2020.

[68] J. Peng, Z. Chen, Y. Shao, Y. Shen, L. Chen, and J. Cao, "Sancus: Staleness-aware communication-avoiding full-graph decentralized training in large-scale graph neural networks," *Proceedings of the VLDB Endowment*, vol. 15, no. 9, pp. 1937–1950, 2022.

[69] A. Virmaux and K. Scaman, "Lipschitz regularity of deep neural networks: analysis and efficient estimation," in *Advances in Neural Information Processing Systems*, vol. 31. Curran Associates, Inc., 2018.

Appendix

A. *Proof of Lemma 1*

Proof: By using the closed-form solution $\boldsymbol{p}^* = (\boldsymbol{I} + c\boldsymbol{L})^{-1}\boldsymbol{x}$ and the fact that $\boldsymbol{A}^{-1} - \boldsymbol{B}^{-1} = \boldsymbol{B}^{-1}(\boldsymbol{B} - \boldsymbol{A})\boldsymbol{A}^{-1}$, we have:

$$\begin{aligned}\hat{\boldsymbol{p}}^* - \boldsymbol{p}^* &= \left((\boldsymbol{I} + c\hat{\boldsymbol{L}})^{-1} - (\boldsymbol{I} + c\boldsymbol{L})^{-1}\right)\boldsymbol{x}\\ &= (\boldsymbol{I} + c\hat{\boldsymbol{L}})^{-1}(c\hat{\boldsymbol{L}} - c\boldsymbol{L})(\boldsymbol{I} + c\boldsymbol{L})^{-1}\boldsymbol{x}\\ &= c(\boldsymbol{I} + c\hat{\boldsymbol{L}})^{-1}(\hat{\boldsymbol{L}} - \boldsymbol{L})\boldsymbol{p}^*.\end{aligned}$$

From Eq. (6), we can acquire the difference between the raw and approximate Laplacian matrices based on the spectral property:

$$\begin{aligned}\|\boldsymbol{L} - \hat{\boldsymbol{L}}\|_2 = &\sup_{\|\boldsymbol{x}\|=1} \boldsymbol{x}^\top(\boldsymbol{L} - \hat{\boldsymbol{L}})\boldsymbol{x} = \boldsymbol{x}_0^\top(\boldsymbol{L} - \hat{\boldsymbol{L}})\boldsymbol{x}_0\\ &\le \epsilon\boldsymbol{x}_0^\top\boldsymbol{I}\boldsymbol{x}_0 = \epsilon,\end{aligned} \tag{14}$$

where $\|\cdot\|_2$ is the matrix spectral norm, and the supremum is achieved when $\boldsymbol{x} = \boldsymbol{x}_0$.

The distance between $\boldsymbol{p}^*$ and $\hat{\boldsymbol{p}}^*$ follows the consistency of spectral norm $\|\boldsymbol{A}\boldsymbol{x}\| \le \|\boldsymbol{A}\|_2\|\boldsymbol{x}\|$. By substituting Eq. (14) and utilizing the property of spectral norm, we have:

$$\begin{aligned}\|\hat{\boldsymbol{p}}^* - \boldsymbol{p}^*\| &\le c\|(\boldsymbol{I} + c\hat{\boldsymbol{L}})^{-1}\|_2 \cdot \|\hat{\boldsymbol{L}} - \boldsymbol{L}\|_2 \cdot \|\boldsymbol{p}^*\|\\ &= c\epsilon\|\boldsymbol{p}^*\| \cdot \max_i \left\{\frac{1}{\lambda_i(\boldsymbol{I} + c\hat{\boldsymbol{L}})}\right\}\\ &= \frac{c\epsilon\|\boldsymbol{p}^*\|}{1 + c\lambda_1(\hat{\boldsymbol{L}})} = c\epsilon\|\boldsymbol{p}^*\|,\end{aligned}$$

where $\lambda_i(\hat{\boldsymbol{L}})$ denotes the i-th smallest eigenvalue of matrix $\hat{\boldsymbol{L}}$ and $\lambda_1(\hat{\boldsymbol{L}}) = 0$. ■

Given the additive nature of the graph specifier, the bound for graph smoothing problem Lemma 1 is dissimilar to the approximation setting with multiplicative similarity bounded by the quadratic form $O(c\epsilon\boldsymbol{p}^\top\boldsymbol{L}\boldsymbol{p})$ [58], [59], but instead correlates with the embedding vector norm $\|\boldsymbol{p}^*\|$. This correlation arises from the bias introduced by the pruned entries in the diffusion matrix, which is associated with the embedding value.

B. *Proof of Theorem 2*

We first derive the following lemma as a general form of Theorem 2 regarding the sparsity and approximation bound without assumptions on entry distribution:

Lemma 5: Given graph $\mathcal{G}$ and embedding $\boldsymbol{p}$, let $\hat{\mathcal{E}}$ be the edge set achieved by graph sparsification with threshold δ_a. The sparsified Laplacian matrix $\hat{\boldsymbol{L}}$ is ϵ-similar to $\boldsymbol{L}$ when $q_a\delta_a \le \epsilon\|\boldsymbol{p}\|$, where q_a is the number of edges removed.

Proof: We outline the diffusion by the general graph adjacency $\boldsymbol{T} = \boldsymbol{A}$. The entry-wise difference matrix for the sparsified diffusion can be derived as $\boldsymbol{\Upsilon} = \boldsymbol{A} - \hat{\boldsymbol{A}} = \hat{\boldsymbol{L}} - \boldsymbol{L}$. Additionally, the pruned edges form the complement set $\mathcal{E}_\Upsilon = \mathcal{E} \setminus \hat{\mathcal{E}}$, and the number of removed edges is $q_a = |\mathcal{E}_\Upsilon|$. If an edge is pruned $(u, v) \in \mathcal{E}_\Upsilon$, the entry $\boldsymbol{\Upsilon}[u, v] = \boldsymbol{A}[u, v]$.

For a current embedding $\boldsymbol{p}$, its product with the difference matrix $\boldsymbol{\Upsilon}\boldsymbol{p}$ only correlates with entries that have been pruned. Based on the Minkowski inequality and sparsification scheme Eq. (9), the L_1 norm of the product vector satisfies:

$$\begin{aligned}\|\boldsymbol{\Upsilon}\boldsymbol{p}\|_1 &= \sum_{u\in\mathcal{V}}\left|\sum_{v\in\mathcal{N}_\Upsilon(u)} \boldsymbol{A}[u, v]\boldsymbol{p}[v]\right| = \sum_{u\in\mathcal{V}}\left|\sum_{v\in\mathcal{N}_\Upsilon(u)} \tau[u, v]\right|\\ &\le \sum_{u\in\mathcal{V}}\sum_{v\in\mathcal{N}_\Upsilon(u)}\left|\tau[u, v]\right| \le \sum_{u\in\mathcal{V}}\sum_{v\in\mathcal{N}_\Upsilon(u)} \delta_a = q_a\delta_a.\end{aligned}$$

Employing the relationship between vector norms given by the Cauchy-Schwarz inequality $\|\boldsymbol{x}\| \le \|\boldsymbol{x}\|_1$, the difference of quadratic forms with regard to graph Laplacian can be bounded as:

$$|\boldsymbol{p}^\top\boldsymbol{L}\boldsymbol{p} - \boldsymbol{p}^\top\hat{\boldsymbol{L}}\boldsymbol{p}| = |\boldsymbol{p}^\top\boldsymbol{\Upsilon}\boldsymbol{p}| \le \|\boldsymbol{p}\| \cdot \|\boldsymbol{\Upsilon}\boldsymbol{p}\| \le q_a\delta_a\|\boldsymbol{p}\|.$$

Referring to Definition 2, Eq. (6) is equivalent to:

$$\left|\boldsymbol{x}^\top(\boldsymbol{L} - \hat{\boldsymbol{L}})\boldsymbol{x}\right| \le \epsilon \cdot \|\boldsymbol{x}\|^2, \quad \forall\boldsymbol{x} \in \mathbb{R}^n. \tag{15}$$

Hence, when $q_a\delta_a \le \epsilon\|\boldsymbol{p}\|$ is met, the sparsified $\hat{\boldsymbol{L}}$ is spectrally similar to $\boldsymbol{L}$ with approximation rate ϵ. ■

Below we show that exact value of q_a is dependent on the actual distribution of entry values. As a reasonable assumption, we particularly investigate the scale-free graph, which is common in realistic large-scale graphs. In this case, the distribution of nodes follows the power law $P(d) = d^{-\alpha}$, where $P(d)$ is the fraction of nodes of degree d for large values, and α is a constant typically within range $2 < \alpha < 3$. We also consider the Gaussian distribution of input attribute values. We hereby provide the proof of Theorem 2:

Proof: Recall in Eq. (9) pruning, there is $\mathcal{E}_\Upsilon = \{(u, v) \mid |\tau[u, v]| \le \delta_a\}$. The fraction of entries below the threshold is thence expressed by the probability:

$$\begin{aligned}q_a &= |\mathcal{E}_\Upsilon| = m \cdot P(\boldsymbol{A}[u, v] \cdot |\boldsymbol{p}[v]| \le \delta_a)\\ &= 2m\int_0^\infty P(\boldsymbol{A}[u, v] \le \frac{\delta_a}{x}) \cdot f_{p[v]}(x)dx,\end{aligned}$$

where $f_{p[v]}(x)$ represents the probability distribution function of embedding values.

Acknowledging the above assumptions, for the normalized adjacency matrix $\tilde{\boldsymbol{A}} = \boldsymbol{D}^{-1/2}\bar{\boldsymbol{A}}\boldsymbol{D}^{-1/2}$, its entry distribution also follows the power law as $x^{-\alpha}$. The embedding value $p[v] \sim N(0, \sigma^2)$. By substituting the distributions we have:

$$\begin{aligned}q_a &= 2m\int_0^\infty \left[1 - \left(\frac{\delta_a}{xx_{min}}\right)^{-\alpha+1}\right]\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{x^2}{2\sigma^2}\right)dx\\ &= 2m(\frac{1}{2} - \int_0^\infty \left(\frac{\delta_a}{xx_{min}}\right)^{-\alpha+1}\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{x^2}{2\sigma^2}\right)dx)\\ &= 2m(\frac{1}{2} - \frac{1}{\sqrt{2\pi}\sigma}\left(\frac{\delta_a}{x_{min}}\right)^{-\alpha+1}\int_0^\infty x^{\alpha-1}\exp\left(-\frac{x^2}{2\sigma^2}\right)dx).\end{aligned}$$

The last term can be given by the Gamma function:

$$\int_0^\infty x^{\alpha-1}\exp\left(-\frac{x^2}{2\sigma^2}\right)dx = \frac{2^{\frac{\alpha}{2}-1}\Gamma\left(\frac{\alpha}{2}\right)}{(\sigma^2)^{\frac{\alpha}{2}}}.$$

Therefore,

$$q_a = m - 2m \frac{2^{\frac{\alpha-3}{2}} \Gamma\left(\frac{\alpha}{2}\right)}{\sqrt{\pi}\sigma^{\alpha+1}} \left(\frac{\delta_a}{x_{min}}\right)^{1-\alpha}, \tag{16}$$

or equivalently, $\eta_a = q_a/m = 1 - C\delta_a^{1-\alpha}$. Therefore, the relative strength of sparsification represented by the threshold $\delta_a/\|\boldsymbol{p}\|$ and the relative sparsity η_a can be represented by each other. Note that when η_a is close to 0, Eq. (16) is less precise since the power law assumption requires statistically sufficient graph components to become meaningful.

Referring to Lemma 5, the approximation bound can be given by:

$$\epsilon = O\left(\eta_a (1-\eta_a)^{\frac{1}{1-\alpha}}\right). \tag{17}$$

Combining Eqs. (16) and (17) leads to to Theorem 2 for specifying the sparsification threshold and bounding the approximation precision, where there is also $t = -1/(1-\alpha) \in [1/3, 1/2]$. ■

C. Proof Sketch of Proposition 3

Consider consecutively sparsifying the diffusion for each hop by Eq. (11). Intuitively, as the edges are gradually removed from the original graph, there is $q_{a,(l_1)} < q_{a,(l_2)}$ for $l_1 < l_2$ with the same relative threshold. If the sparsest graph $\boldsymbol{T}_{(L)}$ satisfies Definition 1, then the multi-layer update is also bounded.

Recall that common GNN learning can be expressed by the graph smoothing process Definition 1. Such optimization problem can be iteratively solved by employing a gradient descent scheme [55], where each iteration derives the l-th hop of graph propagation as in Eq. (4):

$$\boldsymbol{p}_{(l+1)} = \boldsymbol{p}_{(l)} - \frac{b}{2} \cdot \frac{\partial \mathcal{L}}{\partial \boldsymbol{p}}\bigg|_{\boldsymbol{p}=\boldsymbol{p}_{(l)}} = (1-b)\boldsymbol{p}_{(l)} - bc\boldsymbol{L}\boldsymbol{p}_{(l)} + b\boldsymbol{x}. \tag{18}$$

where $b/2$ is the step size and initially there is $\boldsymbol{p}_{(0)} = \boldsymbol{x}$. Eq. (18) is expressive to represent various propagation operations in decoupled GNN models. For example, APPNP [54] can be achieved by letting $b = \alpha$ and $c = (1-\alpha)/\alpha$, while SGC [51] is the edge case with only the graph regularization term.

Now consider the layer-wise graph sparsification under Eq. (18) updates. For the initial state, there is $\boldsymbol{p}_{(0)} = \hat{\boldsymbol{p}}_{(0)} = \boldsymbol{x}$. If in the l-th hop, Eq. (9) edge sparsification is applied to the graph $\boldsymbol{L}$, then the approximation gap is:

$$\boldsymbol{p}_{(l+1)} - \hat{\boldsymbol{p}}_{(l+1)} = (1-b)(\boldsymbol{p}_{(l)} - \hat{\boldsymbol{p}}_{(l)}) + bc(\hat{\boldsymbol{L}}\hat{\boldsymbol{p}}_{(l)} - \boldsymbol{L}\boldsymbol{p}_{(l)}). \tag{19}$$

To demonstrate that $\hat{\boldsymbol{p}}_{(l+1)}$ is an ϵ-approximation, we use induction by assuming $\hat{\boldsymbol{p}}_{(l)} - \boldsymbol{p}_{(l)} = \boldsymbol{\Delta}_p$, $\|\boldsymbol{\Delta}_p/\boldsymbol{p}_{(l)}\| \sim O(\epsilon)$. Then:

$$\begin{aligned}\|\boldsymbol{p}_{(l+1)} - \hat{\boldsymbol{p}}_{(l+1)}\| &= \|-(1-b)\boldsymbol{\Delta}_p + bc(\hat{\boldsymbol{L}}\boldsymbol{p}_{(l)} + \hat{\boldsymbol{L}}\boldsymbol{\Delta}_p - \boldsymbol{L}\boldsymbol{p}_{(l)})\| \\ &\leq \|(bc\boldsymbol{L} - (1-b)\boldsymbol{I})\|_2 \|\boldsymbol{\Delta}_p\| + bc\|\hat{\boldsymbol{L}} - \boldsymbol{L}\|_2 \|\boldsymbol{p}_{(l)} + \boldsymbol{\Delta}_p\| \\ &\leq O(\epsilon) \cdot \|\boldsymbol{p}_{(l)}\| + bc\epsilon(1 + O(\epsilon))\|\boldsymbol{p}_{(l)}\|,\end{aligned}$$

where the inequalities follows from the property of matrix spectral norm and Eq. (14). Hence, the relative error of approximate representation $\hat{\boldsymbol{p}}_{(l+1)}$ is constrained by $O(\epsilon)$.

D. Proof Sketch of Proposition 4

To apply the approximation analysis to iterative GNNs, we first extend the analysis to multi-feature input matrix. Since graph operations among feature dimensions are mutually independent, conclusion from Theorem 2 and Proposition 3 are still valid in their matrix forms. In iterative models, the gradient update similar to Eq. (18) is instead employed to the representation matrix $\boldsymbol{H}_{(l)}$ and derives each layer as:

$$\boldsymbol{P}_{(l)} = (1-b)\boldsymbol{H}_{(l)} - bc\boldsymbol{L}\boldsymbol{H}_{(l)} + b\boldsymbol{X}.$$

As detailed in [55], [56], the update scheme above is able to describe an array of iterative GNNs. For instance, when $\boldsymbol{H}_{(l)} = \boldsymbol{X}$ and $b = 1/c$, it yields the GCN propagation $\boldsymbol{P}_{(l)} = \tilde{\boldsymbol{A}}\boldsymbol{H}_{(l)}$.

To interpret the effect of UNIFEWS sparsification, we first investigate the entry-wise graph pruning in Algorithm 2 and its outcome, i.e., the embedding matrix $\hat{\boldsymbol{P}}_{(l)}$. For simplicity, we assume the sparsification is only employed upon the $(l+1)$-th layer and $\hat{\boldsymbol{H}}_{(l)} = \boldsymbol{H}_{(l)}$. Invoking Eq. (14) and the fact that $\|\boldsymbol{A}\boldsymbol{B}\|_F \leq \|\boldsymbol{A}\|_2 \|\boldsymbol{B}\|_F$, the margin of approximate embeddings can be written as:

$$\|\hat{\boldsymbol{P}}_{(l)} - \boldsymbol{P}_{(l)}\|_F \leq bc\|\hat{\boldsymbol{L}} - \boldsymbol{L}\|_2 \, \|\boldsymbol{H}_{(l)}\|_F \leq bc\epsilon\|\boldsymbol{H}_{(l)}\|_F.$$

Then, consider the weight pruning Eq. (13). The entry-wise error is a composition of joint embedding and weight approximation:

$$\hat{\boldsymbol{\omega}}_{(l+1)}[j,i] - \boldsymbol{\omega}_{(l+1)}[j,i] = \hat{\boldsymbol{W}}_{(l)}[j,i]\hat{\boldsymbol{P}}_{(l)}[:,j] - \boldsymbol{W}_{(l)}[j,i]\boldsymbol{P}_{(l)}[:,j].$$

Let $\boldsymbol{M}_{(l)} = \boldsymbol{W}_{(l)} - \hat{\boldsymbol{W}}_{(l)}$. Recalling the iterative update scheme Eq. (12), the total difference on linear transformation $\boldsymbol{H}' = \boldsymbol{P}_{(l)}\boldsymbol{W}_{(l)}$ is built up by:

$$\hat{\boldsymbol{H}}' - \boldsymbol{H}' = \sum_{i,j=1}^{f} \big(\hat{\boldsymbol{P}}_{(l)}[:,j] - \boldsymbol{P}_{(l)}[:,j]\big)\boldsymbol{W}_{(l)}[j,i] + \hat{\boldsymbol{P}}_{(l)}[:,j]\boldsymbol{M}_{(l)}[j,i].$$

Its first term corresponds to the embedding approximation:

$$\boldsymbol{\Delta}_P \leq \|\hat{\boldsymbol{P}}_{(l)} - \boldsymbol{P}_{(l)}\|_F \, \|\boldsymbol{W}\|_F \leq bc\epsilon\|\boldsymbol{H}_{(l)}\|_F \|\boldsymbol{W}\|_F,$$

and the second term adheres to weight sparsification as per Eq. (13):

$$\boldsymbol{\Delta}_W \leq \sum_{i=1}^{f}\sum_{j=1}^{f} \|\hat{\boldsymbol{P}}_{(l)}[:,j]\boldsymbol{M}_{(l)}[j,i]\| \leq q_w\delta_w,$$

where q_w is the number of pruned weight entries. Finally, the representation matrix in the $(l+1)$-th layer can be bounded by:

$$\|\hat{\boldsymbol{H}}_{(l+1)} - \boldsymbol{H}_{(l+1)}\|_F \leq \ell_\sigma bc\epsilon\|\boldsymbol{H}_{(l)}\|_F \|\boldsymbol{W}\|_F + \ell_\sigma q_w \delta_w, \tag{20}$$

where ℓ_σ is the Lipschitz constant representing the non-linearity of activation function σ [69]. The above equation corresponds to the one in Proposition 4.

The above analysis shows that UNIFEWS with unified graph and weight pruning produces a good approximation of the learned representations across GNN layers, and the margin of output representations is jointly bounded by the graph sparsification rate ϵ and weight threshold δ_w. A recent work [37] offers a theoretical evaluation specifically on model weight pruning throughout training iterations, under more narrow assumptions and the particular GCN scheme. We believe their results could be supplemental to our theory whose focus is the graph perspective.