Graph Elimination Networks

Anonymous Authors[1]

Abstract

Graph Neural Networks (GNNs) excel at capturing local dependencies in graph-structured data but often struggle to model long-range dependencies. Graph Transformers (GTs) mitigate this limitation using global self-attention mechanisms; however, their quadratic complexity with respect to the number of nodes limits scalability. To address these challenges, we propose Graph Elimination Networks (GENs), a novel approach based on Message Passing Neural Networks (MPNNs) that efficiently capture long-range dependencies by enabling implicit node communication within the receptive field. Specifically, GEN introduces a dual self-attention mechanism that computes edge-level and hop-level attention, allowing nodes to adaptively select neighbors based on distance and path information. MPNNs propagation cumulatively aggregates node features across hops, making hop-specific distinctions harder and potentially obscuring hop-level attention. To overcome this, GENs employ an elimination-based propagation that discards features from previous hops, preserving hop-specific distinctions. Unlike GTs, GEN achieves similar performance without requiring additional structures to encode graph information and reduces time complexity to $\mathcal{O}(|E| + |V|)$. Experimental results on multiple benchmark datasets show that GENs consistently outperform or match state-of-the-art methods while significantly improving computational efficiency. Our code is available at URL.

1. Introduction

Graph Neural Networks (GNNs) (Wu et al., 2020) have emerged as powerful tools for handling non-Euclidean graph data, finding wide-ranging applications in social networks (Bian et al., 2020), recommender systems (Fan et al., 2019), bioinformatics (Zhao et al., 2021; Klicpera et al., 2021; Li et al., 2022), chemistry (Do et al., 2019; Yang et al., 2021), and physics (Sanchez-Gonzalez et al., 2020). Traditional GNNs predominantly adhere to the Message-Passing Neural Networks (MPNNs) (Gilmer et al., 2017), which iteratively aggregates neighborhood information through stacked layers to model node dependencies. While computationally efficient, MPNNs suffer from inherent limitations of over-smoothing (Jin et al., 2022; Chen et al., 2020a; Zhang et al., 2021) and over-squashing (Zhang et al., 2022; Black et al., 2023; Park et al., 2023), which degrade their capacity to capture long-range interactions.

Graph Transformers (GTs) (Ying et al., 2021; Chen et al., 2022a; Luo et al., 2022; Rampášek et al., 2022) partially address these issues by employing global attention mechanisms to model all-pairs node dependencies. However, their quadratic complexity relative to the number of nodes renders them prohibitively expensive for large-scale graphs. Approximation methods like Exphormer (Shirzad et al., 2023) , VCR-Graphormer (Fu et al., 2024) and SGFormer (Wu et al., 2024) reduce complexity via sparse attention or linear attention alternatives, yet empirical studies (Gu & Dao, 2023; Wang et al., 2024) suggest these strategies may fail to fully capture genuine long-range dependencies.

This dilemma motivates our exploration of low-cost alternatives that approximate GT-style global interactions. A straightforward approach involves applying Transformer computations to each node's k-hop subgraphs to avoid global operations. However, this requires preprocessing subgraphs for all nodes, leading to an approximate complexity of $\mathcal{O}(|V| \times (\bar{d})^{2k})$, which is generally impractical. Another direction seeks to enhance MPNNs by integrating hop-level attention (Lee et al., 2023). For example, JKNet (Xu et al., 2018b) focuses on mixing multi-hop features, while DAGNN (Liu et al., 2020) and GPRGNN (Chien et al.) apply self-attention to multi-round propagated features. Additionally, MAGNA (Wang et al., 2020) and NAGphormer (Chen et al., 2022b) aggregate contextual information via diffusion. Unfortunately, these methods suffer from feature homogenization (Rusch et al., 2023): the blending of representations across propagation layers progressively erases hop-specific distinctions, ultimately leading to feature over-smoothing.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

To address these challenges, we propose Graph Elimination Networks (GENs), a novel method grounded in two key innovations. First, GENs employ an elimination-based propagation strategy, which is implemented using the Graph Elimination Algorithm (GEA), to discard node information from previous hops at each step, thereby isolating hop-specific features and preventing homogenization. Second, GENs integrate a dual self-attention mechanism comprising edge-level attention (which reweights neighbor importance akin to GAT (Veličković et al., 2017)) and hop-level attention (which dynamically prioritizes information from distinct topological distances). These components synergistically enable GENs to simulate GT-style global interactions within local subgraphs, avoiding explicit subgraph enumeration. At the same time, by leveraging MPNNs' ability to preserve topological distances, GENs also eliminate the need for handcrafted positional encodings, all while maintaining $\mathcal{O}(|E| + |V|)$ complexity. Extensive experiments demonstrate that GENs match or surpass state-of-the-art GTs on both node-level and graph-level benchmarks. Notably, GENs achieve significant computational efficiency gains while maintaining competitive accuracy, particularly excelling on large-scale sparse graphs. Our contributions are as follows:

- We propose the GEA, applied to the propagation stage of MPNNs. After each propagation step, previously observed features are discarded, retaining only newly acquired information. This approach ensures a clear separation of multi-hop features within a node's receptive field, laying the foundation for the dual self-attention mechanism and opening new avenues for GNN architecture design.
- We introduce GENs, an advanced MPNN method that implements edge-level and hop-level attention through elimination-based propagation. This approach achieves implicit GT-style computation on local subgraphs with $\mathcal{O}(|E| + |V|)$ complexity, eliminating positional encoding requirements while enhancing long-range dependency modeling.
- Through extensive experiments, we demonstrate that GENs achieve performance comparable to state-of-the-art GTs across multiple benchmarks, while exhibiting substantial computational efficiency gains particularly on large-scale sparse graphs.

2. Related Work

2.1. Message Passing Neural Networks

Message Passing Neural Networks (MPNNs) (Gilmer et al., 2017) have become a foundational paradigm in graph machine learning, as they iteratively aggregate and propagate information among nodes efficiently. Due to their effectiveness, MPNNs are widely used, with notable implementations such as GAT (Veličković et al., 2017), GCN (Kipf & Welling, 2016; Chen et al., 2020c), GatedGCN (Bresson & Laurent, 2017), and GIN (Xu et al., 2018a). Despite their popularity, MPNNs face several well-known limitations. First, their expressive power is often constrained by the 1-WL isomorphism test (Xu et al., 2018a), making it challenging to distinguish certain classes of non-isomorphic graphs. Second, they are prone to the over-smoothing problem (Li et al., 2018; Oono & Suzuki, 2019; Rusch et al., 2023), where node embeddings become indistinguishable when the network depth increases. Third, they are susceptible to over-squashing (Alon & Yahav, 2020; Akansha, 2023), wherein information from distant nodes becomes compressed and loses granularity in deeper layers.

In response, researchers have proposed various improvements, including higher-order GNNs (Morris et al., 2019; 2020), deeper GNNs (Li et al., 2019; 2020; Rong et al., 2019; Zhang et al., 2022), adaptive propagation methods (Park et al., 2023; Finkelshtein et al., 2023), and graph rewiring strategies (Gutteridge et al., 2023; Choi et al., 2024), all aimed at enhancing the representational capacity of MPNNs while alleviating the aforementioned issues. Interestingly, recent studies (Tönshoff et al., 2023; Yu et al., 2024) have re-evaluated the performance of MPNNs and GTs on benchmark datasets such as LRGB (Dwivedi et al., 2022) and OGB (Hu et al., 2020a), suggesting that the previously claimed advantages of GTs may have been overstated.

2.2. Graph Transformers

Graph Transformers (GTs) represent a new wave of graph learning architectures that integrate the transformer's self-attention mechanism with graph-specific inductive biases. Unlike conventional transformers (Vaswani et al., 2017) designed for Euclidean or sequence data with fixed order, GTs incorporate structural information inherent in graphs through various modifications. Early works, such as GPT-GNN (Hu et al., 2020b), combine graph pooling and unpooling to capture multi-scale relationships, while Graphormer (Ying et al., 2021) employs spectral graph theory to encode global structure using eigenvectors of the normalized Laplacian matrix. Parker et al. (Park et al., 2022) extend this idea via singular value decomposition (SVD), offering an alternative means of integrating global context. Similarly, GraphiT (Mialon et al., 2021; Tambe et al., 2021; Hussain et al., 2022) leverages local edge biases to capture intricate topological details, and hybrid approaches (Chen et al., 2022a) combine traditional GNNs with transformer modules (Cai & Lam, 2020; Ahmad et al., 2021) to address the lack of positional information.

Despite these advancements, one of the main drawbacks of

GTs remains their computational complexity, particularly for large graphs. Sparse attention approaches (Zaheer et al., 2020; Shirzad et al., 2023) reduce the computational burden by limiting the scope of each node's attention, while local attention mechanisms (e.g., NAGphormer (Chen et al., 2022b)) confine attention to neighborhood regions. Low-rank approximations (Rampášek et al., 2022) further reduce the cost of full self-attention by projecting node representations into a lower-dimensional space. More recently, methods like NodeFormer (Wu et al., 2022) and SGFormer (Wu et al., 2024) adopt linear approximations of global self-attention, aiming to retain its benefits without incurring the quadratic complexity inherent in full-attention schemes.

2.3. Graph Mamba

In pursuit of linear-complexity solutions for large-scale graph tasks, Mamba (Gu & Dao, 2023) has gained attention for its specialized self-attention mechanism, which can be viewed as a form of Recurrent Neural Network (RNN) (Grossberg, 2013). It employs a state-space model (SSM) enhanced with self-attention (Aoki, 2013), using the HiPPO algorithm (Gu et al., 2020) to approximate historical input trajectories. By storing relevant information in hidden states, Mamba mitigates the exponential memory decay of earlier RNNs, reducing it to polynomial complexity. Its selective attention mechanism further preserves essential features, offering a memory management strategy similar to transformers. With its adaptability and linear computational complexity, Mamba is a promising candidate for large-scale graph modeling.

The GMN framework (Behrouz & Hashemi, 2024) first introduced a general approach for designing Mamba GNNs. Graph-Mamba (Wang et al., 2024) provides a practical solution by integrating MPNNs with Mamba, replacing the attention module in GraphGPS. Additionally, Chen et al. (Chen et al., 2024) showed that combining deep random walks with Mamba achieves performance comparable to state-of-the-art GTs across various datasets. However, Mamba's reliance on sequential inference increases training time compared to MPNNs with parallel updates. Moreover, the HiPPO algorithm still faces stability and forgetting issues (Yu et al., 2024), requiring further investigation to validate Mamba's efficiency and reliability in large-scale graph applications.

3. Graph Elimination Algorithm

In traditional MPNNs, stacking multiple layers repeatedly propagates known features within a node's receptive field, causing features from different hop distances to become entangled. This not only introduces redundancy into node representations but also complicates the separation of features corresponding to distinct hop distances for each node, thereby hindering the implementation of our dual self-attention mechanism. To address this, we propose the Graph Elimination Algorithm (GEA), which synchronously computes and eliminates redundant terms during each propagation round. By ensuring that only previously unencountered features are incorporated, GEA achieves the disentanglement of features across different hops without compromising the expansion of the GNN's receptive field as the network deepens. This section provides a detailed exposition of GEA, encompassing its conceptual foundation and the specific computational procedures involved.

3.1. Notations

We begin by defining the essential notation used throughout this section. Consider an undirected graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. The node feature matrix is denoted by $X \in \mathbb{R}^{|V| \times F}$, with each node $i \in V$ associated with a feature vector X_i. The neighborhood of node i is represented by $\mu(i)$, the set of its direct neighbors, while D denotes the degree matrix, with d_i being the degree of node i, i.e., the number of edges incident to it. The average node degree is defined as $\bar{d} = \frac{1}{|V|} \sum_{i \in V} d_i$.

In MPNNs, the computation typically proceeds in two phases: Propagation and Update. For instance, in a GCN, the propagation phase can be expressed as $H^{(l)} = \widetilde{A} H^{(l-1)}$ for $1 \leq l \leq l_{\max}$, where $\widetilde{A} = (D + I)^{-1/2}(A + I)(D + I)^{-1/2}$ is the symmetrically normalized adjacency matrix augmented with self-loops, l is the layer index, and $l_{\max}$ is the total number of layers. The update phase is given by $H^{(l)} = \sigma(H^{(l-1)} W^{(l)})$, where $W^{(l)}$ is the weight matrix at layer l and σ is the activation function. Combining these steps yields the standard form $H^{(l)} = \sigma(\widetilde{A} H^{(l-1)} W^{(l)})$. For clarity, we adopt a pointwise perspective and define $C_{ij} = d_i^{1/2} d_j^{1/2}$, allowing the update for node i to be rewritten as:

$$h_i^{(l)} = \sigma\Big(W^{(l)}\big(C_{ii} h_i^{(l-1)} + \sum_{j \in \mu(i) - i} C_{ij} h_j^{(l-1)}\big)\Big). \quad (1)$$

3.2. Derivation of Graph Elimination Algorithm

As exemplified by Eq.(1), traditional MPNNs, such as GCNs, expand a node's receptive field by iteratively aggregating representations from its direct neighbors. While this approach offers linear computational efficiency, it also results in the repeated propagation of known features, leading to redundancy in node representations. To elucidate this, we expand Eq.(1). Define $\mathcal{T}^{(l)}(x) = \sigma(W^{(l)} x)$; for layers $l \geq 2$, the aggregated feature for node i can be expressed as:

$$\sum_{j \in \mu(i) - i} C_{ij} h_j^{(l-1)} = \sum_{j \in \mu(i) - i} C_{ij} \mathcal{T}^{(l-1)}\Big(C_{jj} h_j^{(l-2)} + C_{ji} h_i^{(l-2)} + \sum_{k \in \mu(j) - j - i} C_{jk} h_k^{(l-2)}\Big). \quad (2)$$

Here, $h_i^{(l-1)}$ already encapsulates information from the $l-1$-hop neighbors of node i. However, when aggregating

$h_j^{(l-1)}$ from neighbor j, features from previously encountered nodes—such as i itself or j—are reintroduced, resulting in redundancy. Prior work by (Chen et al., 2022c) attributes this redundancy to inherent graph properties, including undirected edges, backtracking paths, and self-loops, which allow nodes to be revisited during propagation.

To enable hop-specific attention and eliminate the redundancy induced by these edges, we introduce the GEA. Within a fixed GNN propagation process, the repetition caused by such edge types is inherently computable. GEA's strategy involves calculating the redundant contributions introduced by these edges for each node i after every propagation round and subtracting them to neutralize the redundancy. For instance, in the context of Eq.(2), the redundancy term is defined as:

$$R_{i,j}^{(l-1)} = \sum_{j \in \mu(i)-i} C_{ij} \mathcal{T}^{(l-1)} \left(C_{jj} h_j^{(l-2)} + C_{ji} h_i^{(l-2)} \right). \tag{3}$$

Direct subtraction of this term to eliminate redundancy is infeasible due to the presence of nonlinear activation functions σ and trainable weight matrices W between layers. To circumvent this, we impose two constraints: (i) the activation function is fixed as $\sigma(x) = \mathrm{ReLU}(x) = \max(0, x)$; (ii) all initial features $h^{(0)}$ share a uniform sign, and within each column of the weight matrix W, parameters maintain consistent signs (though signs may differ across columns). Under these conditions, the updated feature for node i becomes:

$$\begin{aligned} h_i^{(l)} &= \mathcal{T}^{(l)} \Big(C_{ii} h_i^{(l-1)} - R_{i,j}^{(l-1)} + \sum_{j \in \mu(i)-i} C_{ij} h_j^{(l-1)} \Big) \\ &= \mathcal{T}^{(l)} \Big(C_{ii} h_i^{(l-1)} + \sum_{j \in \mu(i)-i} C_{ij} \mathcal{T}^{(l-1)} \sum_{k \in \mu(j)-j-i} C_{jk} h_k^{(l-2)} \Big). \end{aligned} \tag{4}$$

By recursively expanding Eq.(1), we derive a general form for the redundancy term, with details provided in Appendix A.1. The result is given directly as:

$$\begin{aligned} R_{i,j}^{(1)} &= C_{ij} \mathcal{T}^{(1)} \left(C_{jj} h_j^{(0)} + C_{ji} h_i^{(0)} \right), \\ R_{i,j}^{(2)} &= C_{ij} \mathcal{T}^{(2)} \Big(C_{jj} h_j^{(1)} + C_{ji} h_i^{(1)} - C_{ji} \mathcal{T}^{(1)} (C_{ii} h_i^{(0)} + C_{ij} h_j^{(0)}) \Big), \\ &\cdots \\ R_{i,j}^{(l)} &= C_{ij} \mathcal{T}^{(l)} \Big(C_{jj} h_j^{(l-1)} + C_{ji} h_i^{(l-1)} - R_{j,i}^{(l-1)} \Big), \end{aligned} \tag{5}$$

where $R_{i,j}^{(0)} = 0$. Notably, GEA achieves the objective outlined at the outset of this section: at each propagation step, node representations are updated solely using features from neighbors at the corresponding hop distance. However, this relies on the stringent constraints introduced above, which inevitably limit its practical applicability. This observation motivates further exploration into adapting MPNN architectures to accommodate these constraints while enhancing generalizability.

4. Methodology

Building upon the GEA proposed in Section 3, we introduce GENs, an enhanced architecture rooted in the MPNN framework. GENs leverage GEA to implement elimination-based propagation, enabling the execution of dual self-attention while circumventing GEA's stringent constraints through the introduction of a decoupling mechanism (Spinelli et al., 2020). This ensures compatibility with most conventional MPNN methods. In this section, we elaborate on the implementation of GENs, partitioning it into two primary stages: Elimination-Based Propagation and Update, as illustrated in Figure 1. At the end of this section, we analyze the performance of GENs.

4.1. Elimination-Based Propagation

GENs introduce a decoupled propagation mechanism that separates Message Passing from Feature Transformation (Chen et al., 2020b; Gasteiger et al., 2018), by placing the linear layer in the Update stage instead of incorporating it into the Propagation stage. This decoupling allows multiple rounds of propagation within a single layer without involving the activation function σ or the weight matrix W, ensuring that the GEA conditions are satisfied. Specifically, within a single layer of GENs, GEA is applied iteratively, eliminating redundant terms after each round to prevent previously observed features from being reintroduced.

Let $Z^{(L)}$ denote the node representation matrix updated after the L-th layer. Define $h^{(L,k)}$ as the result of the k-th round of propagation in layer L, with the initial state given by $h^{(L,0)} = Z^{(L-1)}$. Here, we use L to represent the current layer of GENs and K to denote the maximum number of propagation rounds within a single layer. Since the following computations do not involve variables outside layer L, we abbreviate $h^{(L,k)}$ as $h^{(k)}$. The computation of redundant terms in the k-th round of propagation is then given by:

$$R_{ij}^{(k-1)} = C_{ij}^{(k)} (C_{jj}^{(k-1)} h_j^{(k-2)} + C_{ji}^{(k-1)} h_i^{(k-2)} - R_{ji}^{(k-2)}), \tag{6}$$

where $R^{(0)} = 0$. It is important to note that $C_{ij}^{(k)}$ can take any real number, allowing GENs to be compatible with GCN, GAT, or other GNNs. Typically, we obtain it by computing edge-level attention, which, when combined with hop-level attention in the update stage, forms a dual self-attention selection mechanism that simulates the application of GT methods. The computation for edge-level attention is given by:

$$\begin{aligned} e_{ij}^{(k)} &= \mathrm{LeakyReLU} \left(\mathbf{a}^{(k)} \left[h_i^{(k-1)} || h_j^{(k-1)} \right] \right), \\ C_{ij}^{(k)} &= \mathrm{softmax} \left(e_{ij}^{(k)} \right), \end{aligned} \tag{7}$$

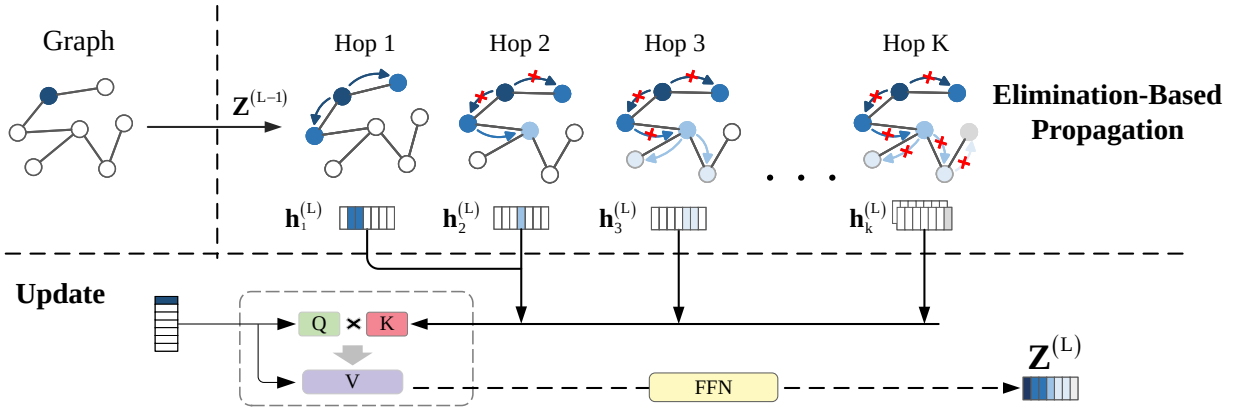

Figure 1. Schematic diagram of the L-th layer of GENs. The network consists of two stages: Elimination-Based propagation and Update. In the Elimination-Based Propagation stage, node features undergo K rounds of propagation, with the GEA applied in each round to distinguish and capture features from neighbors at different hop distances. In the Update stage, self-attention is computed between the node and its K-hop neighbors, and features are extracted through a FFN to form new node representations. The initial features are represented using one-hot embeddings.

where $\mathbf{a}^{(k)} \in \mathbb{R}^{2\times F}$ is the learnable parameter used to compute the edge attention score. The elimination-based propagation stage is defined as:

$$h_i^{(k)} = C_{ii}^{(k)} h_i^{(k-1)} + \sum_{j\in u(i)-i} C_{ij}^{(k)} \left(h_j^{(k-1)} - R_{ij}^{(k-1)} \right). \tag{8}$$

After completing K rounds of propagation, the node representations at different distances are combined into a matrix:

$$H_i = \text{concat}\left(h_i^{(1)}, h_i^{(2)}, \ldots, h_i^{(K)} \right) \in \mathbb{R}^{K\times F}, \tag{9}$$

which contains the final node representations from all rounds of propagation.

4.2. Update

The update layer in GENs is relatively simple and primarily serves to transform the results of multiple rounds of propagation into the output of the hidden layer. H_i contains the aggregated representations of node i at different hops. We learn hop-wise attention through a Transformer encoder, allowing node i to autonomously select the importance of neighbors at different hops. The self-attention module projects the node features and H_i into three subspaces: $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$, then computes the output matrix:

$$\mathbf{Q} = Z_i^{(L-1)} W^Q, \mathbf{K} = H_i W^K, \mathbf{V} = H_i W^V$$
$$\hat{H}_i = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d^K}} \right) \mathbf{V}, \tag{10}$$

where W^Q, W^K, and W^V are learnable parameter matrices. Hop-level attention captures the correlation between node i and nodes at different distances within its receptive field. Combined with edge attention, this is equivalent to constructing a tree that describes a subgraph of the receptive field, with node i as the root. The hop-level attention adaptively selects the relevant layers of neighboring nodes, while the edge-level attention adaptively selects the paths leading to each neighbor node in that layer. Together, these two attentions allow for precise selection of any neighbor within the receptive field. Thus, dual attention implicitly enables direct information exchange between node i and any node in its receptive field, without incurring quadratic costs. Finally, after a residual connection, the result of the dual self-attention is fed into a feed-forward network (FFN), yielding the updated latent space representation:

$$Z_i^{(L)} = \text{FFN}\left(Z_i^{(L-1)} W_o^{(L)} + \hat{H}_i^{(L)} \right). \tag{11}$$

All previous equations ignore the L-th layer for simplicity.

4.3. Performance Analysis

In this section, we analyze the performance of GENs from the perspectives of time complexity and graph topology embedding. This analysis helps in better understanding the advantages of GENs, particularly in terms of computational efficiency and scalability.

Time Complexity. The time complexity of MPNN-based methods is typically $\mathcal{O}(|E|+|V|)$, where $|E|$ represents the number of edges and $|V|$ the number of nodes in the graph.

The additional complexity in GENs arises due to the GEA. According to Eq. (6) and Eq. (7), the elimination terms are computed concurrently with message passing. The total number of direct neighbors across all nodes is proportional to the number of edges $|E|$, with C_{ij} being a variable and h_i being an F-dimensional feature vector. Furthermore, the maximum number of propagation rounds is denoted as K. As a result, the time complexity for computing the elimination terms is $\mathcal{O}(K \times |E| \times F)$. Since both K and F are constants, the overall complexity simplifies to $\mathcal{O}(|E|)$. Additionally, edge attention introduces further computational cost. From Eq. (10) and Eq. (11), we observe that its time complexity is $\mathcal{O}(3 \times |V| \times F^2 + K \times |V| \times F)$, which simplifies to $\mathcal{O}(|V|)$ when constant factors are disregarded. In summary, the overall time complexity of GEN remains $\mathcal{O}(|E| + |V|)$. For sparse graphs, where the number of edges is much smaller than $|V|^2$, GEN's computational cost is significantly lower compared to standard GTs.

Graph Topological Information. The effective learning of GNNs relies on graph topological information, and these models must inherently satisfy permutation invariance (Dufter et al., 2022). However, embedding this topological information into the network presents a challenge. In recent GTs, one common approach is to first use traditional GNNs or random walks to capture the graph structure, and then treat this structure as positional information for global self-attention mechanisms. While effective, this approach introduces additional computational overhead and complicates the process of hyperparameter tuning. In contrast, GENs take a more efficient approach by inheriting the benefits of the MPNN framework. GENs implicitly embed positional information within the receptive field subgraph of each node (Xu et al., 2018a; Maron et al., 2019; Geerts & Reutter, 2022). For instance, suppose the edge attention for a node's 1-hop neighbors is denoted by C_{ij}. For the 2-hop neighbors, the edge attention becomes $C_{ij}C_{jk}$, which naturally decays as the distance increases, given that $C \geq 1$. This implicit embedding of topological information enables GENs to be transferred to most tasks without any graph-structure preprocessing while still delivering competitive results. In summary, GENs combine outstanding versatility with computational efficiency, giving them excellent scalability and adaptability across diverse graph structures without introducing unnecessary computational overhead.

5. Experiments

In this section, we evaluate the performance of GENs on various benchmark datasets from different domains with long-range dependencies and medium scale. In addition, we evaluate the computational efficiency of GENs and compare them with the current state-of-the-art GT methods to verify their effectiveness.

5.1. Experimental Setup

Datasets and Models. We evaluated GENs on three task categories: (1) long-range dependencies (Long-Range Graph Benchmark, LRGB (Dwivedi et al., 2022)), (2) large-scale tasks (Open Graph Benchmark datasets, OGB (Hu et al., 2020a)), and (3) small-scale scenarios (Cora/CiteSeer/PubMed (Sen et al., 2008), Photo (Shchur et al., 2018), WikiCS (Mernyei & Cangea, 2020)) under the protocol (Luo et al., 2024b). We compare GENs against the following baselines: (1) classic MPNN methods, such as GCN (Kipf & Welling, 2016), GIN (Xu et al., 2018a), GAT (Veličković et al., 2017), GraphSAGE (Hamilton et al., 2017) and Gated-GCN (Bresson & Laurent, 2017); (2) state-of-the-art Graph Transformers, including Graphormer (Ying et al., 2021), SAT (Chen et al., 2022a), GPS (Rampášek et al., 2022), NAGphormer (Chen et al., 2022b), GOAT (Kong et al., 2023), NodeFormer (Wu et al., 2022), Exphormer (Shirzad et al., 2023), Drew (Gutteridge et al., 2023), IPR-MPNN (Qian et al., 2024), SGFormer (Wu et al., 2024) and Polynormer (Deng et al., 2024); (3) recent Mamba-based works, such as Graph-Mamba (Wang et al., 2024) and GMN (Behrouz & Hashemi, 2024).

Settings. For different benchmarks, we follow the official evaluation protocols. For example, in the *OGB* and *LRGB* benchmarks, we adhere to the 500K parameter budget constraint and adopt evaluation metrics consistent with those used in prior work. For node classification datasets, we follow the data splits and evaluation metrics provided by (Akiba et al., 2019). Detailed dataset statistics and metric definitions are provided in the corresponding result tables. For all tasks, we report the mean and standard deviation of test performance corresponding to the model checkpoint that achieves the best validation score. All experiments are conducted on a single NVIDIA A100 80 GB GPU. For more experimental details, please refer to Appendix B

5.2. Performance on Long-Range Dependency

We first evaluate GENs' ability to capture long-range dependencies using the widely adopted LRGB datasets. As shown in Table 1, across five LRGB datasets, GENs with dual self-attention mechanisms achieve performance comparable to state-of-the-art Mamba-based methods as well as GT models, securing top performance on three tasks, second-best results on one, and outperforming the best traditional GNN by 5.2% on the remaining task. Notably, tasks where GENs achieved strong performance were also domains where traditional GNNs like GCN and GIN performed relatively well, while GENs substantially narrowed the performance gap with GTs on tasks where GT models dominate. Specifically, GENs lagged behind standard GPS by 4.5% on the COCO-SP task but achieved 21% improvement over GCN, whereas on the Peptides-Func task, they outperformed GPS by 6.1%

Table 1. Test performance on Long-Range Graph Benchmark. The best result is highlighted in bold, and the second and third best results are underlined.

Dataset	PascalVOC-SP	COCO-SP	Peptides-Func	Peptides-Struct	PCQM-Contact
# Graphs	11.4K	123.3K	15.5K	15.5K	529.4K
Avg. # Nodes	479.4	476.9	150.9	150.9	30.1
Avg. # Edges	2,710.5	2,693.7	307.3	307.3	61.0
Metric	F1 ↑	F1 ↑	AP ↑	MAE ↓	MRR ↑
GCN	0.2078 ± 0.0031	0.1338 ± 0.0007	0.6860 ± 0.0050	0.2460 ± 0.0007	0.3424 ± 0.0007
GIN	0.2718 ± 0.0054	0.2125 ± 0.0009	0.6621 ± 0.0067	0.2473 ± 0.0010	0.3509 ± 0.0006
GatedGCN	0.3880 ± 0.0040	0.2922 ± 0.0018	0.6765 ± 0.0047	0.2477 ± 0.0009	0.3495 ± 0.0010
SAT	0.3230 ± 0.0039	0.2592 ± 0.0158	0.6384 ± 0.0121	0.2683 ± 0.0043	-
GPS	0.4440 ± 0.0065	0.3884 ± 0.0055	0.6534 ± 0.0091	0.2509 ± 0.0010	0.3498 ± 0.0005
NAGphormer	0.4006 ± 0.0061	0.3458 ± 0.0070	-	-	-
Exphormer	0.3975 ± 0.0037	0.3455 ± 0.0009	0.6527 ± 0.0043	0.2484 ± 0.0012	**0.3637 ± 0.0020**
Drew	0.3314 ± 0.0024	-	0.7150 ± 0.0044	0.2536 ± 0.0015	0.3444 ± 0.0017
IPR-MPNN	-	-	0.7210 ± 0.0039	0.2462 ± 0.0007	0.3516 ± 0.0102
NBA	-	0.3969 ± 0.0027	0.7206 ± 0.0028	0.2424 ± 0.0010	-
Graph-Mamba	0.4191 ± 0.0126	0.3960 ± 0.0175	0.6739 ± 0.0087	0.2478 ± 0.0016	0.3395 ± 0.0013
GMN	0.4393 ± 0.0112	**0.3974 ± 0.0101**	0.7071 ± 0.0083	0.2473 ± 0.0025	-
GENs	**0.4653 ± 0.0097**	0.3442 ± 0.0139	**0.7142 ± 0.0071**	**0.2432 ± 0.0023**	0.3520 ± 0.0010

Table 2. Results of the ablation study on the Long-Range Graph Benchmark, evaluating the effects of positional encodings (RWSE, LapPE), edge attention (ELA), hop-level attention (HLA), and the graph elimination algorithm (GEA). The best result is shown in bold.

Dataset	PascalVOC-SP F1 ↑	COCO-SP F1 ↑	Peptides-Func AP ↑	Peptides-Struct MAE ↓	PCQM-Contact MRR ↑
GENs	0.4427 ± 0.0131	0.3415 ± 0.0017	0.6860 ± 0.0050	0.2490 ± 0.0016	0.3471 ± 0.0007
GENs+RWSE	**0.4653 ± 0.0097**	0.3404 ± 0.0018	0.6621 ± 0.0067	0.2521 ± 0.0013	0.3440 ± 0.0008
GENs+LapPE	0.4523 ± 0.0162	**0.3442 ± 0.0025**	0.6765 ± 0.0047	0.2436 ± 0.0023	**0.3520 ± 0.0010**
GEN w/o HLA	0.4191 ± 0.0128	0.3301 ± 0.0138	0.6384 ± 0.0121	0.2472 ± 0.0009	0.3480 ± 0.0007
GEN w/o ELA	0.4101 ± 0.0122	0.3016 ± 0.0205	0.6534 ± 0.0091	0.2438 ± 0.0010	0.3498 ± 0.0005
GENs w/o GEA	0.4465 ± 0.0107	0.3356 ± 0.0115	-	**0.2432 ± 0.0023**	0.3496 ± 0.0016

while showing a more modest 2.8% gain over GCN.

To evaluate the contribution of each core component in our model, we conducted an ablation study on the LRGB benchmark, with results reported in Table 2. The findings show that positional encodings commonly used in Graph Transformers, such as RWSE and LapPE, enhance GENs performance when appropriately configured, though unsuitable choices may degrade it. The dual self-attention mechanism is essential: removing either edge-level attention (ELA) or hop-level attention (HLA) significantly reduces accuracy, with the removal of ELA having a greater impact. However, on the Peptides-Func dataset, omitting HLA slightly improves performance. In contrast, excluding the GEA module causes a moderate performance decline, though less severe than removing either attention component. On the Peptides-Struct dataset, the model without GEA marginally outperforms the full configuration, but the difference is minimal.

5.3. Performance on Small-Scale Scenarios

e further evaluated GENs in small-to-medium-scale real-world scenarios, with results shown in Table 3. Contrary to expectations from prior reports, advanced GT methods like SGFormer did not outperform well-tuned traditional GNNs (Luo et al., 2024b). GENs achieved a modest improvement of around 0.5–1% over GCN, yet consistently delivered the best performance across all tasks. These findings suggest that despite being GT approximations, GENs perform competitively with traditional GNNs in small-to-medium-scale settings, demonstrating their versatility in such graph domains.

Table 3. Node classification results over homophilous graphs in small-scale real-world scenarios (%). Comparison of GENs and baseline methods, with the best result highlighted in bold, second and third best results underlined.

Dataset	Cora	CiteSeer	PubMed	Photo	WikiCS
# nodes	2,708	3,327	19,717	7,650	11,701
# edges	5,278	4,732	44,324	119,081	216,123
Metric	Accuracy ↑	Accuracy ↑	Accuracy ↑	Accuracy ↑	Accuracy ↑
GCN	<u>85.10 ± 0.67</u>	<u>73.14 ± 0.50</u>	<u>81.12 ± 0.22</u>	96.10 ± 0.46	80.30 ± 0.62
GraphSAGE	83.88 ± 0.65	72.26 ± 0.55	79.72 ± 0.50	<u>96.78 ± 0.23</u>	<u>80.69 ± 0.31</u>
GAT	84.46 ± 0.55	72.22 ± 0.84	80.28 ± 0.12	96.60 ± 0.33	<u>81.07 ± 0.54</u>
GraphGPS	83.87 ± 0.96	72.73 ± 1.23	79.94 ± 0.26	94.89 ± 0.14	78.66 ± 0.49
NAGphormer	80.92 ± 1.17	70.59 ± 0.89	80.14 ± 1.06	96.14 ± 0.16	77.92 ± 0.93
Exphormer	83.29 ± 1.36	71.85 ± 1.11	79.67 ± 0.73	95.69 ± 0.39	79.38 ± 0.82
GOAT	83.26 ± 1.24	72.21 ± 1.29	80.06 ± 0.67	94.33 ± 0.21	77.96 ± 0.03
NodeFormer	82.73 ± 0.75	72.37 ± 1.20	79.59 ± 0.92	93.43 ± 0.56	75.13 ± 0.93
SGFormer	84.82 ± 0.85	72.72 ± 1.15	<u>80.60 ± 0.49</u>	95.58 ± 0.36	80.05 ± 0.46
Polynormer	83.43 ± 0.90	72.19 ± 0.83	79.35 ± 0.73	96.57 ± 0.23	80.26 ± 0.92
GENs	**85.43 ± 0.42**	**73.56 ± 0.46**	**81.94 ± 0.34**	**97.12 ± 0.24**	**81.48 ± 0.46**

Table 4. Efficiency and accuracy (%) comparison of GENs and baselines on OGBN-Arxiv and OGBN-Products during full-batch training. Results are shown for GENs targeting the best cost-effectiveness by reducing model size, as well as Best-GENs targeting optimal performance. The best result is highlighted in bold, and 'OOM' indicates out-of-memory issues.

Dataset	Metric	GCN	NAGphormer	Exphormer	Graph-Mamba	Ours	
						GENs	Best-GENs
OGBN-Arxiv	Train Time (s)	**0.10**	3.08	1.20	11.90	0.29	1.49
169 K # N	Mem (GB)	**2.73**	6.24	31.89	5.95	7.34	65.02
1,166 K # E	Accuracy	71.69	70.13	72.28	71.78	72.76	**73.51**
OGBN-Products	Train Time (s)	3.12	**1.68**	OOM	736.00	3.83	6.09
2.45 M # N	Mem (GB)	**41.27**	60.09	OOM	63.55	41.65	58.13
61.86 M # E	Accuracy	75.64	73.96	OOM	74.77	77.11	**79.44**

5.4. Cost Efficiency Evaluation

To assess the computational efficiency of GEN, we compared it with state-of-the-art baselines (NAGphormer, Exphormer, and Graph-Mamba), as detailed in Table 4. The results reveal notable differences in scalability and computational cost. Graph-Mamba, for instance, incurs excessive runtime on the ogbn-products dataset under full-batch training, requiring approximately 200 times longer per epoch than standard GCN and over 60 hours for full training. Despite its linear memory scaling, akin to sparse GCN, its high computational demand limits its practicality for large-scale tasks. Conversely, GEN offers training speed and memory usage comparable to GCN, while consistently outperforming it. This advantage becomes more evident with model scaling (Best-GENs), underscoring GEN's superior scalability.

6. Conclusion

In this paper, we introduce GENs as a method that integrates insights from Graph Transformers into the MPNN framework. Addressing the high computational cost of Graph Transformers and the limitations of traditional GNNs in capturing long-range dependencies, GENs leverage graph elimination algorithms and decoupling strategies to efficiently manage multi-hop data. By combining edge-level and hop-level attention, GENs enhance the modeling of complex long-range relationships. Our experiments show that GENs achieve performance comparable to Graph Transformers on small-scale graphs and exhibit significant advantages on large-scale tasks, all while maintaining computational efficiency. These contributions highlight GENs as a promising approach for scalable and efficient graph-based machine learning, offering new directions for future research.

References

Ahmad, W. U., Peng, N., and Chang, K.-W. Gate: graph attention transformer encoder for cross-lingual relation and event extraction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 12462–12470, 2021.

Akansha, S. Over-squashing in graph neural networks: A comprehensive survey. *arXiv preprint arXiv:2308.15568*, 2023.

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. Optuna: A next-generation hyperparameter optimization framework. In *The 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2623–2631, 2019.

Alon, U. and Yahav, E. On the bottleneck of graph neural networks and its practical implications. *arXiv preprint arXiv:2006.05205*, 2020.

Aoki, M. *State space modeling of time series*. Springer Science & Business Media, 2013.

Behrouz, A. and Hashemi, F. Graph mamba: Towards learning on graphs with state space models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 119–130, 2024.

Bian, T., Xiao, X., Xu, T., Zhao, P., Huang, W., Rong, Y., and Huang, J. Rumor detection on social media with bi-directional graph convolutional networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 549–556, 2020.

Black, M., Wan, Z., Nayyeri, A., and Wang, Y. Understanding oversquashing in GNNs through the lens of effective resistance. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 2528–2547. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/black23a.html.

Bresson, X. and Laurent, T. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*, 2017.

Cai, D. and Lam, W. Graph transformer for graph-to-sequence learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7464–7471, 2020.

Chen, D., Lin, Y., Li, W., Li, P., Zhou, J., and Sun, X. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 3438–3445, 2020a.

Chen, D., O'Bray, L., and Borgwardt, K. Structure-aware transformer for graph representation learning. In *International Conference on Machine Learning*, pp. 3469–3489. PMLR, 2022a.

Chen, D., Schulz, T. H., and Borgwardt, K. Learning long range dependencies on graphs via random walks, 2024. URL https://arxiv.org/abs/2406.03386.

Chen, J., Gao, K., Li, G., and He, K. Nagphormer: A tokenized graph transformer for node classification in large graphs. In *The Eleventh International Conference on Learning Representations*, 2022b.

Chen, M., Wei, Z., Ding, B., Li, Y., Yuan, Y., Du, X., and Wen, J.-R. Scalable graph neural networks via bidirectional propagation. *Advances in Neural Information Processing Systems*, 33:14556–14566, 2020b.

Chen, M., Wei, Z., Huang, Z., Ding, B., and Li, Y. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*, pp. 1725–1735. PMLR, 2020c.

Chen, R., Zhang, S., Li, Y., et al. Redundancy-free message passing for graph neural networks. *Advances in Neural Information Processing Systems*, 35:4316–4327, 2022c.

Chien, E., Peng, J., Li, P., and Milenkovic, O. Adaptive universal generalized pagerank graph neural network. In *International Conference on Learning Representations*.

Choi, J., Park, S., Wi, H., Cho, S.-B., and Park, N. Panda: Expanded width-aware message passing beyond rewiring. *arXiv preprint arXiv:2406.03671*, 2024.

Deng, C., Yue, Z., and Zhang, Z. Polynormer: Polynomial-expressive graph transformer in linear time. In *The Twelfth International Conference on Learning Representations*, 2024.

Do, K., Tran, T., and Venkatesh, S. Graph transformation policy network for chemical reaction prediction. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 750–760, 2019.

Dufter, P., Schmitt, M., and Schütze, H. Position information in transformers: An overview. *Computational Linguistics*, 48(3):733–763, 2022.

Dwivedi, V. P., Rampášek, L., Galkin, M., Parviz, A., Wolf, G., Luu, A. T., and Beaini, D. Long range graph benchmark. *Advances in Neural Information Processing Systems*, 35:22326–22340, 2022.

Dwivedi, V. P., Joshi, C. K., Luu, A. T., Laurent, T., Bengio, Y., and Bresson, X. Benchmarking graph neural

networks. *Journal of Machine Learning Research*, 24 (43):1–48, 2023.

Fan, W., Ma, Y., Li, Q., He, Y., Zhao, E., Tang, J., and Yin, D. Graph neural networks for social recommendation. In *The World Wide Web Conference*, pp. 417–426, 2019.

Finkelshtein, B., Huang, X., Bronstein, M., and Ceylan, I. I. Cooperative graph neural networks. *arXiv preprint arXiv:2310.01267*, 2023.

Fu, D., Hua, Z., Xie, Y., Fang, J., Zhang, S., Sancak, K., Wu, H., Malevich, A., He, J., and Long, B. Vcr-graphormer: A mini-batch graph transformer via virtual connections. In *The Twelfth International Conference on Learning Representations*, 2024.

Gasteiger, J., Bojchevski, A., and Günnemann, S. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018.

Geerts, F. and Reutter, J. L. Expressiveness and approximation properties of graph neural networks. *arXiv preprint arXiv:2204.04661*, 2022.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pp. 1263–1272. PMLR, 2017.

Grossberg, S. Recurrent neural networks. *Scholarpedia*, 8 (2):1888, 2013.

Gu, A. and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Gu, A., Dao, T., Ermon, S., Rudra, A., and Ré, C. Hippo: Recurrent memory with optimal polynomial projections. *Advances in neural information processing systems*, 33: 1474–1487, 2020.

Gutteridge, B., Dong, X., Bronstein, M. M., and Di Giovanni, F. Drew: Dynamically rewired message passing with delay. In *International Conference on Machine Learning*, pp. 12252–12267. PMLR, 2023.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems*, 30, 2017.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. Open graph benchmark: Datasets for machine learning on graphs. *Advances in Neural Information Processing Systems*, 33:22118–22133, 2020a.

Hu, Z., Dong, Y., Wang, K., Chang, K.-W., and Sun, Y. Gpt-gnn: Generative pre-training of graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 1857–1867, 2020b.

Hussain, M. S., Zaki, M. J., and Subramanian, D. Global self-attention as a replacement for graph convolution. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 655–665, 2022.

Jin, W., Liu, X., Ma, Y., Aggarwal, C., and Tang, J. Feature overcorrelation in deep graph neural networks: A new perspective. *arXiv preprint arXiv:2206.07743*, 2022.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Klicpera, J., Becker, F., and Günnemann, S. Gemnet: Universal directional graph neural networks for molecules. *arXiv e-prints*, pp. arXiv–2106, 2021.

Kong, K., Chen, J., Kirchenbauer, J., Ni, R., Bruss, C. B., and Goldstein, T. Goat: A global transformer on large-scale graphs. In *International Conference on Machine Learning*, pp. 17375–17390. PMLR, 2023.

Lee, S. Y., Bu, F., Yoo, J., and Shin, K. Towards deep attention in graph neural networks: Problems and remedies. In *International Conference on Machine Learning*, pp. 18774–18795. PMLR, 2023.

Li, G., Muller, M., Thabet, A., and Ghanem, B. Deepgcns: Can gcns go as deep as cnns? In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9267–9276, 2019.

Li, G., Xiong, C., Thabet, A., and Ghanem, B. Deepergcn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739*, 2020.

Li, Q., Han, Z., and Wu, X.-M. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Li, S., Zhou, J., Xu, T., Dou, D., and Xiong, H. Geomgcl: Geometric graph contrastive learning for molecular property prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 4541–4549, 2022.

Liu, M., Gao, H., and Ji, S. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 338–348, 2020.

Luo, S., Li, S., Zheng, S., Liu, T.-Y., Wang, L., and He, D. Your transformer may not be as powerful as you expect. *Advances in Neural Information Processing Systems*, 35: 4301–4315, 2022.

Luo, Y., Shi, L., and Wu, X.-M. Classic gnns are strong baselines: Reassessing gnns for node classification. *arXiv preprint arXiv:2406.08993*, 2024a.

Luo, Y., Shi, L., and Wu, X.-M. Classic GNNs are strong baselines: Reassessing GNNs for node classification. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024b. URL https://openreview.net/forum?id=xkljKdGe4E.

Maron, H., Fetaya, E., Segol, N., and Lipman, Y. On the universality of invariant networks. In *International Conference on Machine Learning*, pp. 4363–4371. PMLR, 2019.

Mernyei, P. and Cangea, C. Wiki-cs: A wikipedia-based benchmark for graph neural networks. *arXiv preprint arXiv:2007.02901*, 2020.

Mialon, G., Chen, D., Selosse, M., and Mairal, J. Graphit: Encoding graph structure in transformers. *arXiv preprint arXiv:2106.05667*, 2021.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 4602–4609, 2019.

Morris, C., Rattan, G., and Mutzel, P. Weisfeiler and leman go sparse: Towards scalable higher-order graph embeddings. *Advances in Neural Information Processing Systems*, 33:21824–21840, 2020.

Oono, K. and Suzuki, T. Graph neural networks exponentially lose expressive power for node classification. *arXiv preprint arXiv:1905.10947*, 2019.

Park, S., Ryu, N., Kim, G., Woo, D., Yun, S.-Y., and Ahn, S. Non-backtracking graph neural networks. *arXiv preprint arXiv:2310.07430*, 2023.

Park, W., Chang, W.-G., Lee, D., Kim, J., et al. Grpe: Relative positional encoding for graph transformer. In *ICLR2022 Machine Learning for Drug Discovery*, 2022.

Qian, C., Manolache, A., Morris, C., and Niepert, M. Probabilistic graph rewiring via virtual nodes. *arXiv preprint arXiv:2405.17311*, 2024.

Rampášek, L., Galkin, M., Dwivedi, V. P., Luu, A. T., Wolf, G., and Beaini, D. Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems*, 35:14501–14515, 2022.

Rong, Y., Huang, W., Xu, T., and Huang, J. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903*, 2019.

Rusch, T. K., Bronstein, M. M., and Mishra, S. A survey on oversmoothing in graph neural networks. *arXiv preprint arXiv:2303.10993*, 2023.

Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., and Battaglia, P. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pp. 8459–8468. PMLR, 2020.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Shchur, O., Mumme, M., Bojchevski, A., and Günnemann, S. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

Shirzad, H., Velingker, A., Venkatachalam, B., Sutherland, D. J., and Sinop, A. K. Exphormer: Sparse transformers for graphs. In *International Conference on Machine Learning*, pp. 31613–31632. PMLR, 2023.

Spinelli, I., Scardapane, S., and Uncini, A. Adaptive propagation graph convolutional network. *IEEE Transactions on Neural Networks and Learning Systems*, 32(10): 4755–4760, 2020.

Tambe, T., Hooper, C., Pentecost, L., Jia, T., Yang, E.-Y., Donato, M., Sanh, V., Whatmough, P., Rush, A. M., Brooks, D., et al. Edgebert: Sentence-level energy optimizations for latency-aware multi-task nlp inference. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 830–844, 2021.

Tönshoff, J., Ritzert, M., Rosenbluth, E., and Grohe, M. Where did the gap go? reassessing the long-range graph benchmark. *arXiv preprint arXiv:2309.00367*, 2023.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Wang, C., Tsepa, O., Ma, J., and Wang, B. Graph-mamba: Towards long-range graph sequence modeling with selective state spaces. *arXiv preprint arXiv:2402.00789*, 2024.

Wang, G., Ying, R., Huang, J., and Leskovec, J. Multi-hop attention graph neural network. *arXiv preprint arXiv:2009.14332*, 2020.

Wu, Q., Zhao, W., Li, Z., Wipf, D. P., and Yan, J. Nodeformer: A scalable graph structure learning transformer for node classification. *Advances in Neural Information Processing Systems*, 35:27387–27401, 2022.

Wu, Q., Zhao, W., Yang, C., Zhang, H., Nie, F., Jiang, H., Bian, Y., and Yan, J. Simplifying and empowering transformers for large-graph representations. *Advances in Neural Information Processing Systems*, 36, 2024.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Philip, S. Y. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2020.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018a.

Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.-i., and Jegelka, S. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, pp. 5453–5462. PMLR, 2018b.

Yang, Z., Chakraborty, M., and White, A. D. Predicting chemical shifts with graph neural networks. *Chemical Science*, 12(32):10802–10809, 2021.

Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., and Liu, T.-Y. Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems*, 34:28877–28888, 2021.

Yu, A., Mahoney, M. W., and Erichson, N. B. There is hope to avoid hippos for long-memory state space models. *arXiv preprint arXiv:2405.13975*, 2024.

Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.

Zhang, W., Yang, M., Sheng, Z., Li, Y., Ouyang, W., Tao, Y., Yang, Z., and Cui, B. Node dependent local smoothing for scalable graph learning. *Advances in Neural Information Processing Systems*, 34:20321–20332, 2021.

Zhang, W., Sheng, Z., Yin, Z., Jiang, Y., Xia, Y., Gao, J., Yang, Z., and Cui, B. Model degradation hinders deep graph neural networks. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 2493–2503, 2022.

Zhao, C., Liu, S., Huang, F., Liu, S., and Zhang, W. Csgnn: Contrastive self-supervised graph neural network for molecular interaction prediction. In *IJCAI*, pp. 3756–3763, 2021.

A. Derivation for Elimination Algorithm

A.1. Detailed Derivation

Eliminating redundancy in GNN propagation is a complex challenge. In this section, we provide a detailed explanation of the thought process and derivation behind the elimination algorithm. During GNN propagation, node representations often redundantly aggregate features from previously visited nodes. This occurs because graphs inherently contain paths that allow nodes to be revisited, including undirected edges, backtracking edges, and self-loops. To address this issue, we introduce an elimination algorithm that filters out the redundant information introduced by these edges during the propagation process.

We take GCN as an example to derive the general formula for the elimination algorithm. The complete formula of GCN at the l-th layer is given by:

$$h_i^{(l)} = \sigma\Big(W_l\Big(C_{ii}h_i^{(l-1)} + \sum_{j\in\mu(i)-i} C_{ij}h_j^{(l-1)}\Big)\Big), \quad (1)$$

where $\sigma(\cdot)$ is the nonlinear function between layers, such as the composite function $\mathrm{Relu}(\cdot)$ that consists of the activation function Relu. W_l is the learnable parameter matrix for the l-th layer, C_{ij} is used to normalize the node features, such that $C_{ij} = {d_i}^{-1/2} * {d_j}^{-1/2}$, and d_i denotes the degree of node i, and $h_i^{(0)} = x_i^{(0)}$, and $\mu\,(i)$ is the set of neighborhood nodes of node i.

Since σ and W almost always occur together, we simplify the notation by omitting the parentheses between σ and W in the following equations. The goal of the elimination algorithm is to eliminate the feature redundancy in the propagation process. There is no redundancy in the first layer, but since this formula will be used later, we still give it here:

$$h_i^{(1)} = \sigma W_1\Big(C_{ii}h_i^{(0)} + \sum_{j\in\mu(i)-i} C_{ij}h_j^{(0)}\Big). \quad (2)$$

The second layer is calculated as follows:

$$h_i^{(2)} = \sigma W_2\Big(C_{ii}h_i^{(1)} + \sum_{j\in\mu(i)-i} C_{ij}h_j^{(1)}\Big). \quad (3)$$

We try to rewrite the second term on the right side of the formula, with the goal of splitting the $h_i^{(0)}$ contained in it. At this time, there is:

$$\sum_{j\in\mu(i)-i} C_{ij}h_j^{(1)} = \sum_{j\in\mu(i)-i} C_{ij}\sigma W_1\Big(C_{jj}h_j^{(0)} + C_{ji}h_i^{(0)} + \sum_{k\in\mu(j)-j-i} C_{jk}h_k^{(0)}\Big), \quad (4)$$

Given that $\sigma(\cdot)$ is a nonlinear function, we usually cannot simplify this equation further. However, for the sake of discussion, we assume that $\sigma(\cdot)$ is $\mathrm{Relu} = \max(0, x)$, and that all the initial features $h^{(0)}$ have the same sign and all the parameters in the same column of W have the same sign as well. Under these assumptions, we can split the terms inside the nonlinear function. Note that C_{ij} is always non-negative, so we have:

$$\begin{aligned}\sum_{j\in\mu(i)-i} C_{ij}h_j^{(1)} = &\sum_{j\in\mu(i)-i} C_{ij}W_1 \sum_{k\in\mu(j)-j-i} C_{jk}h_k^{(0)} \\ &+ \sum_{j\in\mu(i)-i} C_{ij}\Big|W_1 \sum_{k\in\mu(j)-j-i} C_{jk}h_k^{(0)}\Big| \\ &+ \sum_{j\in\mu(i)-i} C_{ij}W_1\Big(C_{jj}h_j^{(0)} + C_{ji}h_i^{(0)}\Big) \\ &+ \sum_{j\in\mu(i)-i} C_{ij}\Big(\Big|W_1\Big(C_{jj}h_j^{(0)} + C_{ji}h_i^{(0)}\Big)\Big|\Big),\end{aligned} \quad (5)$$

there $h_i^{(1)}$ in Eq.(3) already contains features around one hop, that the features about 0-hop and 1-hop in the second term are redundant, that is, we need to delete the third and fourth terms from the right side of the equation. For convenience, we denote the deleted terms as:

$$del_1 = \sum_{j\in\mu(i)-i} C_{ij}\sigma W_1\Big(C_{jj}h_j^{(0)} + C_{ji}h_i^{(0)}\Big), \quad (6)$$

so that $h_i^{(2)}$ after elimination can be written as:

$$h_i^{(2)} = \sigma W_2\Big(C_{ii}h_i^{(1)} - del_1 + \sum_{j\in\mu(i)-i} C_{ij}h_j^{(1)}\Big). \quad (7)$$

In addition, according to Eq.(5), we can also get another expression of this formula:

$$h_i^{(2)} = fW_2\Big(C_{ii}h_i^{(1)} + \sum_{j\in\mu(i)-i} C_{ij}\sigma W_1 \sum_{k\in\mu(j)-j-i} C_{jk}h_k^{(0)}\Big). \quad (8)$$

Continue to consider the third layer, we calculate with the same premise, first write out the formula of the third layer:

$$h_i^{(3)} = \sigma W_3\Big(C_{ii}h_i^{(2)} + \sum_{j\in\mu(i)-i} C_{ij}h_j^{(2)}\Big). \quad (9)$$

According to Eq.(8), rewrite the second term in the above formula again:

$$\begin{aligned}\sum_{j\in\mu(i)-i} C_{ij}h_j^{(2)} = &\sum_{j\in\mu(i)-i} C_{ij}\sigma W_2\Big(C_{jj}h_j^{(1)} \\ &+ \sum_{k\in\mu(j)-j} C_{jk}\sigma W_1 \sum_{l\in\mu(k)-k-j} C_{kl}h_l^{(0)}\Big) \\ = &\sum_{j\in\mu(i)-i} C_{ij}\sigma W_2\Big(C_{jj}h_j^{(1)} \\ &+ C_{ji}\sigma W_1 \sum_{l\in\mu(i)-i-j} C_{il}h_l^{(0)} \\ &+ \sum_{k\in\mu(j)-j-i} C_{jk}\sigma W_1 \sum_{l\in\mu(k)-k-j} C_{kl}h_l^{(0)}\Big),\end{aligned} \quad (10)$$

since the activation function Relu guarantees that each term has the same sign, there is:

$$\sum_{j\in\mu(i)-i} C_{ij}h_j^{(2)} = \sum_{j\in\mu(i)-i}\Big(C_{ij}W_2\Big(C_{jj}h_j^{(1)} + C_{ji}\sigma W_1\sum_{l\in\mu(i)-i-j}C_{il}h_l^{(0)} + \sum_{k\in\mu(j)-j-i}C_{jk}\sigma W_1\sum_{l\in\mu(k)-k-j}C_{kl}h_l^{(0)}\Big) + C_{ij}\Big(\Big|W_2\Big(C_{jj}h_j^{(1)} + C_{ji}\sigma W_1\sum_{l\in\mu(i)-i-j}C_{il}h_l^{(0)}\Big)\Big| + \Big|W_2\Big(\sum_{k\in\mu(j)-j-i}C_{jk}\sigma W_1\sum_{l\in\mu(k)-k-j}C_{kl}h_l^{(0)}\Big)\Big|\Big)\Big). \tag{11}$$

The information about two hops around the node in the above formula is unnecessary. It is important to note that the neighborhood of i necessarily contains i itself, whereas the neighborhood of j contains both i and j. However, the neighborhood of k does not have this property, and i may or may not be included in it. Therefore, we only need to delete j and k from the neighborhood of k. When k and i are connected by an edge, there is a loop in the graph, but we cannot know whether this loop exists or not. We can only assume that there is no loop here, and it is similar later. Therefore, **elimination only holds for acyclic graphs**. The deleted terms at this time are:

$$\begin{aligned} del_2 &= \sum_{j\in\mu(i)-i} C_{ij}\Big(W_2\Big(C_{jj}h_j^{(1)} + C_{ji}\sigma W_1\sum_{l\in\mu(i)-i-j}C_{il}h_l^{(0)}\Big) + \Big|W_2\Big(C_{jj}h_j^{(1)} + C_{ji}\sigma W_1\sum_{l\in\mu(i)-i-j}C_{il}h_l^{(0)}\Big)\Big|\Big) \\ &= \sum_{j\in\mu(i)-i} C_{ij}\sigma W_2\Big(C_{jj}h_j^{(1)} + C_{ji}\sigma W_1\sum_{l\in\mu(i)-i-j}C_{il}h_l^{(0)}\Big), \end{aligned} \tag{12}$$

where $h_l^{(0)}$ is essentially a one-hop neighborhood node of i. We transform Eq.(2) as follows:

$$h_i^{(1)} = \sigma W_1\Big(C_{ii}h_i^{(0)} + C_{ij}h_j^{(0)} + \sum_{l\in\mu(i)-i-j}C_{il}h_l^{(0)}\Big), \tag{13}$$

so, there is:

$$C_{ji}\sigma W_1\sum_{l\in\mu(i)-i-j}C_{il}h_l^{(0)} = C_{ji}h_i^{(1)} - C_{ji}\sigma W_1\left(C_{ii}h_i^{(0)} + C_{ij}h_j^{(0)}\right). \tag{14}$$

Substituting Eq.(14) into Eq.(12), we have:

$$del_2 = \sum_{j\in\mu(i)-i} C_{ij}\sigma W_2\Big(C_{jj}h_j^{(1)} + C_{ji}h_i^{(1)} - C_{ji}\sigma W_1\big(C_{ii}h_i^{(0)} + C_{ij}h_j^{(0)}\big)\Big), \tag{15}$$

so that $h_i^{(3)}$ after elimination can be written as:

$$h_i^{(3)} = \sigma W_3\Big(C_{ii}h_i^{(2)} - del_2 + \sum_{j\in\mu(i)-i}C_{ij}h_j^{(2)}\Big). \tag{16}$$

According to Eq.(11), $h_i^{(3)}$ can also be written as:

$$h_i^{(3)} = \sigma W_3\Big(C_{ii}h_i^{(2)} + \sum_{j\in\mu(i)-i}C_{ij}\sigma W_2\sum_{k\in\mu(j)-j-i}C_{jk}\sigma W_1\sum_{l\in\mu(k)-k-j}C_{kl}h_l^{(0)}\Big). \tag{17}$$

Continue to consider the fourth layer, and the corresponding equation is:

$$h_i^{(4)} = \sigma W_4\Big(C_{ii}h_i^{(3)} + \sum_{j\in\mu(i)-i}C_{ij}h_j^{(3)}\Big). \tag{18}$$

Rewrite the above formula using Eq.(17). According to previous discussions, the items of f can be split directly at this point. For simplicity, we directly give the result after splitting:

$$\begin{aligned} \sum_{j\in\mu(i)-i} C_{ij}h_j^{(3)} &= \sum_{j\in\mu(i)-i} C_{ij}\sigma W_3\Big(C_{jj}h_j^{(2)} + \sum_{k\in\mu(j)-j}C_{jk}\sigma W_2\sum_{l\in\mu(k)-k-j}C_{kl}\sigma W_1\sum_{m\in\ \mu(l)-l-k}C_{lm}h_{lm}^{(0)}\Big) \\ &= \sum_{j\in\mu(i)-i} C_{ij}\sigma W_3\Big(C_{jj}h_j^{(2)} + C_{ji}\sigma W_2\sum_{l\in\mu(i)-i-j}C_{il}\sigma W_1\sum_{m\in\ \mu(l)-l-i}C_{lm}h_m^{(0)}\Big) \\ &\quad + \sum_{j\in\mu(i)-i} C_{ij}\sigma W_3\sum_{k\in\mu(j)-j}C_{jk}\sigma W_2 \sum_{l\in\mu(k)-k-j}C_{kl}\sigma W_1\sum_{m\in\ \mu(l)-l-k}C_{lm}h_{lm}^{(0)}. \end{aligned} \tag{19}$$

Transform Eq.(8) as follows:

$$h_i^{(2)} = \sigma W_2\Big(C_{ii}h_i^{(1)} + C_{ij}\sigma W_1\sum_{k\in\mu(j)-j-i}C_{jk}h_k^{(0)} + \sum_{l\in\mu(i)-i-j}C_{il}\sigma W_1\sum_{m\in\mu(l)-l-i}C_{lm}h_m^{(0)}\Big), \tag{20}$$

so, there is:

$$C_{ji}\sigma W_2\sum_{l\in\mu(i)-i-j}C_{il}\sigma W_1\sum_{m\in\mu(l)-l-i}C_{lm}h_m^{(0)} = C_{ji}h_i^{(2)} - \sigma W_2\Big(C_{ii}h_i^{(1)} + C_{ij}\sigma W_1\sum_{k\in\mu(j)-j-i}C_{jk}h_k^{(0)}\Big). \tag{21}$$

Substituting Eq.(21) into Eq.(19), we get that this layer's deleted term is:

$$del_3 = \sum_{j\in\mu(i)-i} C_{ij}\sigma W_3\Big(C_{jj}h_j^{(2)} + C_{ji}h_i^{(2)} - \sigma W_2\big(C_{ii}h_i^{(1)} + C_{ij}\sigma W_1\sum_{k\in\ \mu(j)-j-i}C_{jk}h_k^{(0)}\big)\Big). \tag{22}$$

According to Eq.(2), there is such an equation:

$$h_j^{(1)} = \sigma W_1 \Big(C_{jj} h_j^{(0)} + C_{ji} h_i^{(0)} + \sum_{k \in \mu(j)-j-i} C_{jk} h_k^{(0)} \Big). \tag{23}$$

The acquisition method of Eq.(23) is consistent with that of Eq.(13). The difference is only to replace node i with node j. The subscript of the final term uses j or l to indicate that there is no difference at this point. According to Eq.(23), the deleted term becomes:

$$\begin{aligned} del_3 = \sum_{j \in \mu(i)-i} C_{ij} \sigma W_3 \Big(C_{jj} h_j^{(2)} + C_{ji} h_i^{(2)} - \sigma W_2 \Big(C_{ii} h_i^{(1)} \\ + C_{ij} h_j^{(1)} - C_{ij} \sigma W_1 \big(C_{jj} h_j^{(0)} + C_{ji} h_i^{(0)} \big) \Big) \Big). \end{aligned} \tag{24}$$

so, there is:

$$h_i^{(4)} = \sigma W_4 \Big(C_{ii} h_i^{(3)} - del_3 + \sum_{j \in \mu(i)-i} C_{ij} h_j^{(3)} \Big), \tag{25}$$

and there is:

$$\begin{aligned} h_i^{(4)} = \sigma W_4 \Big(C_{ii} h_i^{(3)} + \sum_{j \in \mu(i)-i} C_{ij} \sigma W_3 \\ \sum_{k \in \mu(j)-j} C_{jk} \sigma W_2 \sum_{l \in \mu(k)-k-j} C_{kl} \sigma W_1 \sum_{m \in \mu(l)-l-k} C_{lm} h_{lm}^{(0)} \Big). \end{aligned} \tag{26}$$

By analogy, comparing Eq.(24), Eq.(15) and Eq.(6), it can be found that any layer's deleted term del can always be simplified to a form expressed by $h_i^{(0)}$ and $h_j^{(0)}$. In order to accurately describe this deleted term, we first define a nested function as follows:

$$\begin{aligned} f_1^E &= C_{ij} \sigma W_1 \left(C_{jj} h_j^{(0)} + C_{ji} h_i^{(0)} \right), \\ f_2^E &= C_{ij} \sigma W_2 \Big(C_{jj} h_j^{(1)} + C_{ji} h_i^{(1)} - C_{ji} \sigma W_1 \left(C_{ii} h_i^{(0)} + C_{ij} h_j^{(0)} \right) \Big), \\ &\cdots \\ f_l^E &= C_{ij} \sigma W_l \Big(C_{jj} h_j^{(l-1)} + C_{ji} h_i^{(l-1)} - C_{ji} \sigma W_{l-1} \Big(C_{ii} h_i^{(l-2)} \\ &\quad + C_{ij} h_j^{(l-2)} - f_{l-2}^E \Big) \Big), \end{aligned} \tag{27}$$

where l>2. At this time, the deleted term of layer $l+1$ can be expressed as:

$$del_l = \sum_{j \in \mu(i)-i} f_l^E. \tag{28}$$

So that GCN eliminates computing generalization at layer $l+1$ as follows:

$$h_i^{(l+1)} = \sigma W_{l+1} \Big(C_{ii} h_i^{(l)} - del_l + \sum_{j \in \mu(i)-i} C_{ij} h_j^{(l)} \Big). \tag{29}$$

This result can be verified by experiment, and the result after eliminating redundancy is shown in Figure 1.

A.2. Algorithm Extension

In this section, we discuss how to extend the elimination algorithm to other propagation methods. For GCN propagation, C_{ij} is the same in each layer, and it is always greater than or equal to 0, but these may not hold in other propagation methods. From Eq.(4), we can see that the nonlinear function σ is the main challenge for generalizing elimination. Without this function, C_{ij} and W would be unrestricted. If the nonlinear function is indispensable, then formula Eq.(5) and all subsequent formulas are only valid when C_{ij} is always positive. However, the result of Eq.(29) is not affected by the variation of C_{ij} across different layers. Assuming that C_{ij} is different in each layer, according to Eq.(27), there must be:

$$\begin{aligned} f_l^E = C_{ij}^{(l+1)} \sigma W_l \Big(C_{jj}^{(l)} h_j^{(l-1)} + C_{ji}^{(l)} h_i^{(l-1)} \\ - C_{ji}^{(l)} \sigma W_{l-1} \Big(C_{ii}^{(l-1)} h_i^{(l-2)} \\ + C_{ij}^{(l-1)} h_j^{(l-2)} - f_{l-2}^E \Big) \Big). \end{aligned} \tag{30}$$

In actual code, it is not very convenient to directly calculate the above equation. We can split it as follows:

$$\begin{aligned} R_{ij}^{(l)} &= C_{ij}^{(l+1)} \sigma W_l \big(C_{jj}^{(l)} h_j^{(l-1)} + C_{ji}^{(l)} h_i^{(l-1)} - R_{ji}^{(l-1)} \big), \\ del_l &= \sum_{j \in \mu(i)-i} R_{ij}^{(l)}, \end{aligned} \tag{31}$$

where $de_0 = dl_0 = 0$, $C_{ij}^{(l)}$ is the propagation coefficient from node j to node i in the l-th layer, usually $C_{ij}^{(l)} \neq C_{ji}^{(l)}$. Eq.(31) is equivalent to Eq.(28), and this result applies to most mainstream GNNs. In addition, when σ or W does not exist, we only need to delete σ or W_l according to Eq.(31).

If we consider the simplest case, that is, assuming that C_{ij} is always 1, and σ and W_3 do not exist, then Eq.(22) can be rewritten as:

$$\begin{aligned} del_3 &= \sum_{j \in \mu(i)-i} \Big(h_j^{(2)} + h_i^{(2)} - \Big(h_i^{(1)} + \sum_{k \in \mu(j)-j-i} h_k^{(0)} \Big) \Big) \\ &= \sum_{j \in \mu(i)-i} \left(h_j^{(2)} - h_i^{(2)} \right) + \sum_{j \in \mu(i)-i} \Big(h_i^{(1)} + \sum_{k \in \ \mu(j)-j-i} h_k^{(0)} \Big). \end{aligned} \tag{32}$$

For the first term on the right side of equation, we can rewrite it as follows:

$$\sum_{j \in \mu(i)-i} \left(h_j^{(2)} + h_i^{(2)} \right) = (d_i - 1)\, h_i^{(2)} + h_i^{(2)} + \sum_{j \in \mu(i)-i} h_j^{(2)}. \tag{33}$$

On the one hand, according to Eq.(16), there is:

$$\sum_{j \in \mu(i)-i} \left(h_j^{(2)} + h_i^{(2)} \right) = (d_i - 1)\, h_i^{(2)} + h_i^{(3)} + del_2. \tag{34}$$

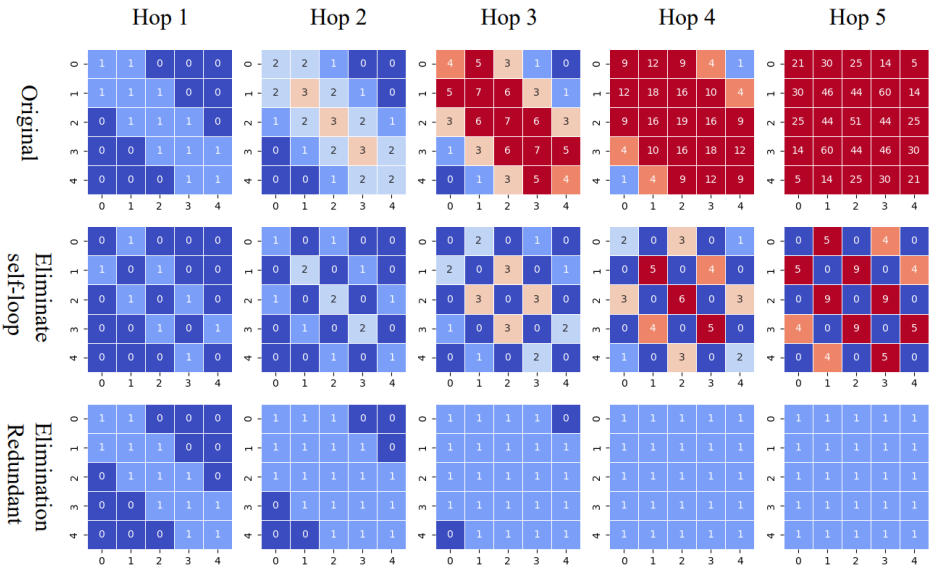

Figure 1. The change of the feature matrix after five times of propagation in the GNNs, assume that the nodes are initially one-hot encoded, ignoring the parameters and normalization coefficients. The upper part is the result of using the graph elimination algorithm, and the lower part is the result of self-loop elimination.

On the other hand, according to Eq.(12), there is:

$$del_2 = \sum_{j\in\mu(i)-i} \left(h_j^{(1)} + \sum_{l\in\mu(i)-i-j} h_l^{(0)}\right). \tag{35}$$

The second term on the right side of Eq.(32) has:

$$\sum_{j\in\mu(i)-i} \left(-\left(h_i^{(1)} + \sum_{k\in\mu(j)-j-i} h_k^{(0)}\right)\right) = -del_2. \tag{36}$$

According to Eq.(36), Eq.(34) and Eq.(32), the final Eq.(25) can be rewritten as:

$$h_i^{(4)} = (1-d_i)\, h_i^{(2)} + \sum_{j\in\mu(i)-i} h_j^{(3)}. \tag{37}$$

The same result applies to Eq.(29), which means that if C_{ij} is constantly 1, and σ and W do not exist, this equation can be simplified as:

$$h_i^{(l+1)} = (1-d_i)\, h_i^{(l-1)} + \sum_{j\in\mu(i)-i} h_j^{(l)}, \tag{38}$$

where $l \geq 1$. When satisfying the above conditions, Eq.(38) is the simplest form to implement elimination calculation.

A.3. Simplification of the Elimination Algorithm

As we observed in the previous section, while the graph elimination algorithm effectively removes feature redundancy in neighborhood aggregation, its computational cost can be prohibitive on large-scale graphs. Therefore, we propose a simpler and more efficient alternative: self-loop elimination. This method removes self-loops from the GNN layers, which means that only nodes at odd or even hop distances in the receptive field are included in the results of odd or even propagation steps, respectively. This reduces the exponential growth of redundant computations from every hop to every two hops, as illustrated in the middle part of Figure 1. Subsequently, we can sum the outputs of all layers or only the last two layers to obtain representations with reduced redundancy. Although the self-loop elimination algorithm is not as thorough as the graph elimination algorithm in eliminating feature redundancy, it offers several advantages: it is easier to implement, generally more convenient to use, and, in some scenarios, more broadly applicable. However, it should be noted that self-loop elimination, like the graph elimination algorithm, is also only suitable for acyclic graphs.

B. Experimental Details and Additional Results

In this section we present the complete experimental settings, data-split protocol, and hyper-parameter search space, and report results that are not shown in the main paper.

B.1. Experimental Setup

Datasets and Tasks We evaluate our model on five public benchmarks: (1) **LRGB** (Dwivedi et al., 2022); (2) **OGB** (Hu et al., 2020a); (3) classical node-classification datasets—Cora, CiteSeer, PubMed (Sen et al., 2008), Photo (Shchur et al., 2018), and WikiCS (Mernyei & Cangea,

Table 1. Node classification results on heterophilous graphs (%). The best result is highlighted in bold, and the second and third best results are underlined.

Dataset	Squirrel	Chameleon	Amazon-Ratings	Roman-Empire	Questions
# nodes	2,223	890	24,492	22,662	48,921
# edges	46,998	8,854	93,050	32,927	153,540
Metric	Accuracy ↑	Accuracy ↑	Accuracy ↑	Accuracy ↑	ROC-AUC ↑
GCN	45.01 ± 1.63	46.29 ± 3.40	53.80 ± 0.00	91.27 ± 0.20	79.02 ± 0.60
GraphSAGE	40.78 ± 1.47	44.81 ± 4.74	55.40 ± 0.21	91.06 ± 0.27	77.21 ± 1.28
GAT	41.73 ± 2.07	44.13 ± 2.41	54.92 ± 0.61	90.63 ± 0.14	77.95 ± 0.51
H2GCN	35.10 ± 1.15	26.75 ± 3.64	36.47 ± 0.23	60.11 ± 0.52	63.59 ± 1.46
CPGNN	30.04 ± 2.03	33.00 ± 3.15	39.79 ± 0.77	63.96 ± 0.62	65.96 ± 1.95
GPRGNN	38.95 ± 1.99	39.93 ± 3.30	44.88 ± 0.34	64.85 ± 0.27	55.48 ± 0.91
FSGNN	35.92 ± 1.32	40.61 ± 2.97	52.74 ± 0.83	79.92 ± 0.56	78.86 ± 2.92
GloGNN	35.11 ± 1.24	25.90 ± 3.58	36.89 ± 0.14	59.63 ± 0.69	65.74 ± 1.19
GraphGPS	39.81 ± 2.28	41.55 ± 3.91	53.27 ± 0.06	82.72 ± 0.08	72.56 ± 1.33
NodeFormer	38.89 ± 2.67	36.38 ± 3.85	43.79 ± 2.57	74.83 ± 0.81	75.02 ± 1.61
SGFormer	42.65 ± 2.41	45.21 ± 3.72	54.14 ± 2.00	80.01 ± 0.44	73.81 ± 0.59
Polynormer	41.97 ± 2.14	41.97 ± 3.18	54.96 ± 2.22	**92.66 ± 0.60**	78.94 ± 0.78
GENs	**46.25 ± 0.61**	**47.10 ± 4.40**	**55.92 ± 0.43**	92.52 ± 0.50	**79.22± 0.74**

2020); (4) **Graph Benchmarks** (Dwivedi et al., 2023) (added in this appendix); (5) five heterogeneous-graph tasks (Luo et al., 2024b): Squirrel, Chameleon, Amazon_Ratings, Roman_Empire, and Questions (also added here).

Data Splits and Evaluation Protocol For benchmarks with official splits—OGB, Graph Benchmarks, and LRGB—we strictly follow the provided train/validation/test partitions. For the node-classification and heterogeneous-graph tasks we reproduce the splits released by (Luo et al., 2024a) in their public repository. All results are reported on the test set at the epoch where the validation performance peaks. Repetition counts are: five runs on LRGB and Graph Benchmarks, ten runs on OGB, and 3–5 runs on the node-classification and heterogeneous-graph tasks, matching Luo *et al.*

Hyper-parameter Search For node-classification and heterogeneous-graph tasks, we perform Bayesian optimisation using OPTUNA (Akiba et al., 2019). The search space is defined as follows: $L \in \{2, \ldots, 12\}$; learning rate $\eta \in [10^{-4}, 10^{-2}]$ (log-uniform); weight decay $w_d \in \{10^{-1}, 10^{-2}, 10^{-3}, 5{\times}10^{-4}, 10^{-4}, 5{\times}10^{-5}, 10^{-5}, 10^{-6}\}$; dropout $p \in [0, 0.9]$ with an interval of 0.1; batch size and hidden dimension are selected from $\{2^4, 2^5, \ldots, 2^{11}\}$; number of attention heads $h \in \{1, \ldots, 8\}$; training epochs $\in \{200, 300, 500, 1000, 1500, 2000, 2500\}$; propagation steps $K \in \{1, \ldots, 8\}$; compression coefficient $\gamma \in [0, 1]$. For the remaining tasks, we start from the default GCN configuration and manually fine-tune within the same ranges. All optimal configurations are available in our repository.

Experimental Environment All experiments are conducted on a single NVIDIA A100-80GB GPU, and no early-stopping strategy is employed. Evaluation metrics and dataset statistics are summarised in the appendix tables.

B.2. Additional Results

In this section, we report results for three additional tasks excluded from the main text due to space limitations: heterogeneous-graph tasks, Graph Benchmarks, and performance comparisons on acyclic graphs. Full results for the heterogeneous graph tasks and Graph benchmarks are detailed in Tables 1 and 2, respectively. GENs surpass state-of-the-art GTs on small-scale heterogeneous graphs and exhibit performance comparable to GT on the medium-scale LRGB benchmark. These findings are consistent with GENs' performance in the main experiments, further underscoring their versatility and robustness across diverse graph-based tasks.

To examine the influence of graph cycles on GENs, we constructed an acyclic subset by extracting all cycle-free samples from the ZINC 250K dataset, resulting in 1,109 samples. This subset was partitioned following the original split, with 950 samples for training, 32 for testing, and 127 for validation.We contrasted these results with the ZINC subset of the Graph benchmark, which comprises 12K samples, of which only 66 (split as 55 training, 10 testing, and 1 validation) are acyclic, effectively representing a predominantly cyclic collection. The outcomes are presented in Table 3. Notably, both GENs and GPS experienced a perfor-

Table 2. Test performance on benchmarks from (Dwivedi et al., 2023). The best result is highlighted in bold, and the second and third best results are underlined.

Dataset	**ZINC**	**MNIST**	**CIFAR10**	**PATTERN**	**CLUSTER**
# Graphs	12K	70K	60K	14K	12K
Avg. # Nodes	23.2	70.6	117.6	118.9	117.2
Avg. # Edges	24.9	564.5	941.1	3039.3	2150.9
Metric	MAE ↓	ACC ↑	ACC ↑	ACC ↑	ACC ↑
GCN	0.367 ± 0.011	90.705 ± 0.218	55.710 ± 0.381	71.892 ± 0.334	68.498 ± 0.976
GIN	0.526 ± 0.051	96.485 ± 0.252	55.255 ± 1.527	85.387 ± 0.136	64.716 ± 1.553
GAT	0.384 ± 0.007	95.553 ± 0.205	56.223 ± 0.455	78.271 ± 0.186	70.587 ± 0.447
GatedGCN	0.282 ± 0.015	97.340 ± 0.143	67.312 ± 0.311	85.568 ± 0.088	73.840 ± 0.326
Graphormer	0.122 ± 0.006	97.905 ± 0.176	65.978 ± 0.579	-	-
SAT	0.089 ± 0.002	-	-	86.848 ± 0.037	77.856 ± 0.104
GPS	**0.070 ± 0.004**	98.051 ± 0.126	72.298 ± 0.356	86.685 ± 0.059	78.016 ± 0.180
Exphormer	0.132 ± 0.005	98.550 ± 0.030	74.690 ± 0.130	86.700 ± 0.030	**78.070 ± 0.037**
Graph-Mamba	0.107 ± 0.003	98.420 ± 0.080	73.700 ± 0.340	86.710 ± 0.050	76.800 ± 0.360
GMN	-	98.390 ± 0.180	75.760 ± 0.420	**87.140 ± 0.120**	-
GENs	0.074 ± 0.012	**98.650 ± 0.106**	**76.231 ± 0.337**	87.021 ± 0.095	77.902 ± 0.097

Table 3. MAE Comparison of GNNs on Acyclic vs. Cyclic subsets of ZINC.

Dataset	**#Samples**	**GENs**	**GAT**	**GCN**	**GraphGPS**	**Exphormer**
ZINC (Acyclic)	1,109	**0.095** ± 0.011	0.151 ± 0.016	0.141 ± 0.012	0.142 ± 0.018	0.174 ± 0.015
ZINC (Cyclic)	12K	0.074 ± 0.012	0.384 ± 0.007	0.367 ± 0.011	**0.070** ± 0.004	0.132 ± 0.005
Abs. Diff.	-	-0.021	+0.233	+0.226	-0.072	-0.042

mance decrease on the acyclic subset relative to the original dataset, whereas traditional GNNs (GCN, GAT, GIN) generally exhibited improved performance. This observation suggests that traditional GNNs may be more sensitive to the presence of cycles compared to GENs, highlighting the latter's robustness in managing cyclic structures.