Two Heads Are Better Than One: Boosting Graph Sparse Training via Semantic and Topological Awareness

Guibin Zhang [1 2 †] **Yanwei Yue** [1 †] **Kun Wang** [3] **Junfeng Fang** [3] **Yongduo Sui** [3] **Kai Wang** [4] **Yuxuan Liang** [2] **Dawei Cheng** [1] **Shirui Pan** [5] **Tianlong Chen** [6]

Abstract

Graph Neural Networks (GNNs) excel in various graph learning tasks but face computational challenges when applied to large-scale graphs. A promising solution is to remove non-essential edges to reduce the computational overheads in GNN. Previous literature generally falls into two categories: topology-guided and semantic-guided. The former maintains certain graph topological properties yet often underperforms on GNNs. The latter performs well at lower sparsity on GNNs but faces performance collapse at higher sparsity levels. With this in mind, we propose a new research line and concept termed **Graph Sparse Training (GST)**, which dynamically manipulates sparsity at the data level. Specifically, GST initially constructs a topology & semantic anchor at a low training cost, followed by performing dynamic sparse training to align the sparse graph with the anchor. We introduce the **Equilibria Sparsification Principle** to guide this process, balancing the preservation of both topological and semantic information. Ultimately, GST produces a sparse graph with maximum topological integrity and no performance degradation. Extensive experiments on 6 datasets and 5 backbones showcase that GST **(I)** identifies subgraphs at higher graph sparsity levels ($1.67\% \sim 15.85\%\uparrow$) than state-of-the-art sparsification methods, **(II)** preserves more key spectral properties, **(III)** achieves $1.27-3.42\times$ speedup in GNN inference and **(IV)** successfully helps graph adversarial defense and graph lottery tickets. The code is available here.

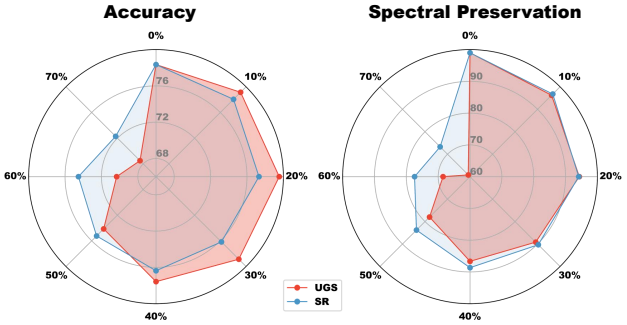

Figure 1. Graph sparsifier (UGS & Spectral Radius) comparison on Ogbn-Proteins using 3-layer GraphSAGE at varying graph sparsity levels $\{10\%, 20\%, \cdots, 60\%\}$. (***Left***) ROC-AUC score after different levels of sparsification. (***Right***) The spectral preservation ratio[2] of the obtained sparse graph.

1. Introduction

Graph Neural Networks (GNNs) (Wu et al., 2020; Zhou et al., 2020) have emerged as leading solutions for graph-related learning tasks (Velickovic et al., 2017; Hamilton et al., 2017; Zhang & Chen, 2018; 2019; Ying et al., 2018b; Fang et al., 2024). The essence of GNNs lies in their iterative neighborhood *aggregation* and *update* processes. The former aggregates neighboring node embeddings via sparse matrix-based operations (*e.g.*, `SpMM` and `SDDMM`), while the latter updates central nodes using dense matrix-based operations (*e.g.*, `MatMul`) (Fey & Lenssen, 2019; Wang et al., 2019). `SpMM` often accounts for the largest portion (50%∼70%) of GNN's computational load (Qiu et al., 2021; Liu et al., 2023), mainly determined by the size of the graph input. However, large-scale graph inputs are commonplace in real-world applications (Wang et al., 2022; Jin et al., 2021; Zhang et al., 2024), presenting significant computational challenges that impede feature aggregation in GNN training and inference. These issues sadly drop a daunting obstacle on the way toward GNNs' on-device deployment, especially in resource-constrained environments.

Given that `SpMM` dominates the computational load of GNNs,

*Equal contribution [1]Tongji Univerity [2]The Hong Kong University of Science and Technology (Guangzhou) [3]University of Science and Technology of China [4]National University of Singapore [5]Griffith University [6]Massachusetts Institute of Technology. Correspondence to: Kun Wang <wk520529@mail.ustc.edu.cu>, Tianlong Chen <tianlong@mit.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

[2]Following (Liu et al., 2023), we define the spectral preservation ratio as the relative error of the top-200 eigenvalues, represented as $\sum_{i=1}^{200} \frac{|\lambda_i - \lambda_i'|}{\lambda_i}$, where λ_i and λ_i' denote the i-th eigenvalue of the original and sparse graphs, respectively.

and the execution time of such sparse operation is proportional to the graph's edge count (Liu et al., 2023), previous approaches to mitigate this inefficiency primarily concentrated on graph sparsification by discarding non-essential edges. They generally fall into two categories:

■ The first research line involves calculating **topology-guided edge scores** for removing the less significant ones, such as resistance effective (Spielman & Srivastava, 2008), eigenvalue (Batson et al., 2013), pairwise distances betweenness (David, 1995), size of all cuts (Benczúr & Karger, 1996) and node degree distributions (Eden et al., 2018).
■ The other primarily relies on **semantic-based scores** derived from gradients, momentum, magnitude, *etc.*, during GNN training, utilizing methods such as trainable masks (Chen et al., 2021; Wang et al., 2023c;a), probabilistic sampling (Zheng et al., 2020; Luo et al., 2021) or meta-gradients (Wan & Schweitzer, 2021) to score and prune non-essential edges.

Scrutinizing the existing graph *sparsifiers*, they focus on either topology or semantics, thereby suffer from inherent limitations correspondingly:

- ***Limited Performance.*** Though theoretically capable of preserving specific spectral properties (Batson et al., 2013), topology-guided sparsifiers overlook the rich graph features, GNN training dynamics, and downstream tasks, resulting in suboptimal performance when integrated with GNN (Luo et al., 2021). Semantic-guided sparsifiers dynamically assess edge importance during GNN training, while they often suffer from significant performance collapse when targeting high graph sparsity, due mostly to detrimental impact on the overall connectivity of the graph (Hui et al., 2023; Wang et al., 2023b).
- ***Empirical Observations.*** We apply a typical semantic-based sparsifier, UGS (Chen et al., 2021) and a topology-based sparsifier, Spectral Radius (SR) (Chan & Akoglu, 2016; Karhadkar et al., 2023) on Ogbn-Proteins (Hu et al., 2020). From Fig. 1, we observe that: ❶ UGS maintains GNN performance well at lower sparsity levels (0%∼30%), yet encounters a dramatic performance drop (over 5.5%↓) at higher sparsity (50%∼), accompanied by significant spectral preservation loss; ❷ SR consistently fails to match GNN performance on sparse graphs with that on dense graphs, though its performance deterioration is gradual, with a milder spectral preservation loss.

The aforementioned observations prompt questions about graph sparsifiers: *Can we ideally leverage the strengths and mitigate the shortcomings of both research lines?* Going beyond this, considering topology-guided methods are inherently ahead of and independent of training, while semantic-guided ones are closely intertwined with GNN optimization, their integration is naturally incongruent. We further question that: *How can we effectively combine topology- and semantic-aware sparsification, embodying the principle that two heads are better than one?*

To this end, we take the first step to explore dynamic graph-level sparse training in both a semantic- and topology-aware manner. We innovatively propose **Graph Sparse Training** (**GST**), which iteratively updates and maintains a sub-counterpart of the original graph during training. To highlight, GST explores the dynamic sparse issue at data-level for the first time, opening a potential pathway for integrating spectral theory and dynamic training algorithms. Additionally, we introduce the **Equilibria Sparsification Principle** to guide the exploration process, which offers a new paradigm for future sparsifier development.

More specifically, considering the inherent properties of the graph itself, GST starts with full graph training to build the dependable anchor, providing a well-aligned semantic and topological benchmark for later dynamic sparse training. Then, GST prunes towards the larger sparsity, resulting in a sparse graph with suboptimal performance (Ma et al., 2021; Frankle et al., 2020; Chen et al., 2021). Finally, the GNN continues to train while iteratively updating the graph, *i.e.*, pruning and growing an equal number of edges, to discover an optimal sparse structure. During each update, we adhere to the **Equilibria Sparsification Principle** that prioritizes both semantic and topological significance, *explicitly minimizing the discrepancy between the current sparse graph and the anchor graph*. We methodically explore the graph structure through GST, aiming to retain the maximum amount of semantic and topological information. Briefly put, our contributions can be summarized as:

- In this work, we systematically review two graph sparsification research lines: **topology-guided** and **semantic-guided**. For the first time, we develop a novel framework that combines their strengths and explicitly preserves graph topology integrity to boost GNN performance maintenance at extreme graph sparsity levels.
- We introduce **Graph Sparse Training** (**GST**), an innovative pruning framework that manipulates sparsity at the data level, exploring sparse graph structure with both semantic- and topology-awareness. Additionally, GST boasts high versatility, effectively aiding various mainstream tasks, including graph adversarial defense and graph lottery tickets.
- Our extensive experiments on 6 datasets and 5 backbones demonstrate that GST **(I)** identifies subgraphs at higher graph sparsity levels ($1.67\% \sim 15.85\% \uparrow$) compared to SOTA sparsification methods in node classification tasks, without compromising performance, **(II)** effectively preserves $\sim 15\%$ more spectral properties, **(III)** achieves tangible $1.27-3.42\times$ GNN inference speedup, and **(IV)** successfully combat edge perturbation ($0.35\% \sim 7.23\%\uparrow$) and enhances graph lottery tickets ($0.22\% \sim 1.58\%\uparrow$).

2. Related Work

Topology-based sparsifiers are early attempts at graph lightweighting. Essentially, they use a theoretically inspired pre-defined metric to score edge importance and prune those with lower scores. SCAN (Xu et al., 2007) assesses global connectivity importance using Jaccard similarity. L-spar (Satuluri et al., 2011) and Local Similarity (Hamann et al., 2016) filter edges locally based on Jaccard similarity. (Spielman & Srivastava, 2008) suggested graph sampling via effective resistance, and (Liu et al., 2023) accelerated its computation with unbiased approximation. However, these methods, typically effective in traditional graph tasks (*e.g.,* graph clustering) (Chen et al., 2023), struggle to maintain performance when applied to GNNs due to their unawareness of GNN training (Luo et al., 2021).

Semantic-based sparsifiers are more closely integrated with GNNs. They aim to dynamically identify important edges via GNN training semantics. NeuralSparse (Zheng et al., 2020) introduces a learning-based sparsification network to select k edges per node. Meta-gradient sparsifier (Wan & Kokel, 2021) leverages meta-gradients to periodically remove edges. Additionally, Graph Lottery Ticket (GLT) (Chen et al., 2021; Wang et al., 2023c; 2022) offers a new paradigm for graph sparsification, which gradually prunes the graph to target sparsity using iterative magnitude pruning (IMP). However, these methods often struggle at higher graph sparsity levels, due to their disruption of the graph topology (Hui et al., 2023).

Sprase Training	Target	Semantic?	Topology?	PRC§
SNIP (Lee et al., 2018)	Sparse NN	✓	✗	✓
RigL (Evci et al., 2020)	Sparse NN	✓	✗	✓
IMDB (Hoang et al., 2023)	Sparse NN	✗	✓	✓
DSnT (Zhang et al., 2023)	Sparse LLM	✓	✗	✓
GST (Ours)	Sparse Graph	✓	✓	✓

§ PRC: Prune Rate Control

Table 1. Comparison among different sparsifiers.

Dynamic Sparse Training (DST) is gaining attention as a method to reduce the computational cost in network training. It entails starting with a randomly sparsified network and periodically updating its connections (Evci et al., 2020). Recent advancements have expanded DST's scope both theoretically (Liu et al., 2021; Huang et al., 2023) and algorithmically (Liu et al., 2020; Zhang et al., 2023). **Our GST fundamentally differs from DST in at least two aspects**. as shown in Tab. 1: (1) *Research target*. While current DST focuses on sparsifying network parameters, GST innovatively explores the feasibility of the "prune and regrow" paradigm in graph data. (2) *Update methodology*. DST essentially aligns with semantic-guided methods, adjusting connections based on semantic information (gradient, momentum, *etc.*) during network training. In contrast, GST considers both semantic and topological information of the graph. We place more discussions and comparisons with related work in Appendix B.

3. Methodology

3.1. Notations and Formulations

Notations. Consider a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V}$ represents the nodes and $\mathcal{E}$ the edges. The feature matrix of $\mathcal{G}$ is $\mathbf{X} \in \mathbb{R}^{N\times D}$, with $N = |\mathcal{V}|$ being the total node count and each node $v_i \in \mathcal{V}$ having an D-dimensional feature vector $\boldsymbol{x}_i = \mathbf{X}[i, \cdot]$. $\mathbf{A} \in \{0, 1\}^{N\times N}$ is the adjacency matrix, where $\mathbf{A}[i, j] = 1$ signifies an edge $(v_i, v_j) \in \mathcal{E}$. Let $f(\cdot; \mathbf{\Theta})$ represent a GNN with $\mathbf{\Theta}$ as its parameters, then $\mathbf{Z} = f(\{\mathbf{A}, \mathbf{X}\}; \mathbf{\Theta})$ is the model output. Note that $\mathcal{G}$ is alternatively denoted as $\{\mathbf{A}, \mathbf{X}\}$ here, and we will interchangeably use its two notations. Consider semi-supervised node classification, its objective function $\mathcal{L}$ is:

$$\mathcal{L}(\mathcal{G}, \mathbf{\Theta}) = -\frac{1}{|\mathcal{V}_{\text{label}}|} \sum_{v_i \in \mathcal{V}_{\text{label}}} y_i \log(z_i), \tag{1}$$

where $\mathcal{L}$ is the cross-entropy loss calculated over the labelled node set $\mathcal{V}_{\text{label}}$, and y_i and z_i denotes the label and prediction of v_i, respectively.

Problem Formulation. The target of graph sparsification is to identify the sparsest subgraph $\mathcal{G}^{\text{sub}} = \{\mathbf{A} \odot \mathbf{M}_g, \mathbf{X}\}$, where $\mathbf{M}_g \in \{0, 1\}^{N\times N}$ is a binary mask indicating edge removal if $\mathbf{M}_g[i, j] = 0$ for $(v_i, v_j) \in \mathcal{E}$ and $\odot$ denotes element-wise product. Additionally, the subgraph $\mathcal{G}^{\text{sub}}$ should satisfy the condition that training the GNN on $\mathcal{G}^{\text{sub}}$ achieves performance comparable to that on the original graph $\mathcal{G}$. The objective is as follows:

$$\begin{aligned} &\arg\max_{\mathbf{M}_g} \; s_g = 1 - \frac{\|\mathbf{M}_g\|_0}{\|\mathbf{A}\|_0} \\ &\text{s. t. } |\mathcal{L}(\{\mathbf{A}, \mathbf{X}\}, \mathbf{\Theta}) - \mathcal{L}(\{\mathbf{A} \odot \mathbf{M}_g, \mathbf{X}\}, \mathbf{\Theta})| < \epsilon, \end{aligned} \tag{2}$$

where s_g is the graph sparsity, $\|\cdot\|_0$ counts the number of non-zero elements, and ϵ is the threshold for permissible performance difference. We define the *extreme graph sparsity* as the maximal s_g without compromising accuracy.

3.2. Framework Overview

Fig. 2 illustrates the overall workflow of GST. Starting with an initialized GNN and the adjacency matrix $\mathbf{A}$ as input, we first train the GNN together with a graph mask $\boldsymbol{m}_g \in \mathbb{R}^{|\mathbf{A}|}$ produced by a graph masker for limited epochs. During this phase, we capture the optimal topological and semantic information as the *anchor graph* (Sec 3.3). Subsequently, we apply one-shot pruning to m_g, reducing it to the desired sparsity $s_g\%$ and creating a sketched sparse graph. From this sparse base, we continue training, dynamically fine-tuning the sparse graph's structure to align semantically and topologically with the *anchor graph* (Sec. 3.4). Through iterative refinement and exploration of graph structure, we ultimately achieve a sparse graph with maintained performance, reduced memory footprint, and faster inference speed (Sec. 3.5).

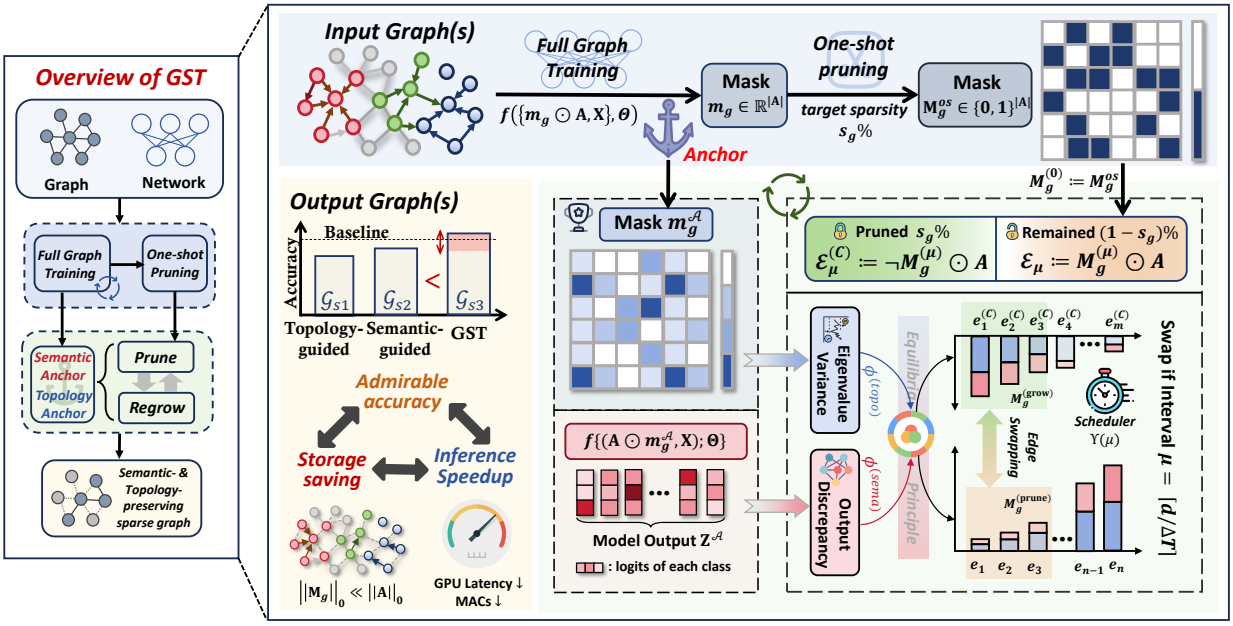

Figure 2. (***Left***) The overview of GST; (***Right***) The detailed pipeline of GST. GST dynamically adjusts and updates the sparse graph, guided by an anchor graph from full-graph training, to optimize topological and semantic preservation, and finally yields a sparse subgraph at the desired sparsity along with admirable accuracy, storage saving, and inference speedup.

3.3. Pursuing Anchor Graph

As outlined above, both topology-guided and semantic-guided sparsifiers typically derive guidance, explicitly or implicitly, from the original dense graph, striving to minimize their divergence from it (Tsitsulin & Perozzi, 2023). In line with the principle of efficiency in DST, several works such as Early-Bird (EB) and Graph EB have demonstrated that limited training can also construct high-quality benchmarks (Achille et al., 2018; You et al., 2019; 2022). Therefore, we propose conducting limited training on the original graph, *i.e.*, full graph training, to capture an *anchor* encompassing the original graph's topology and semantics. This approach provides natural "instructions" for subsequent dynamic adjustment of the target sparse graph. To start with, we derive a (dense) graph mask $\boldsymbol{m}_g \in \mathbb{R}^{|\mathbf{A}|}$ with a parameterized graph masker Ψ:

$$\boldsymbol{m}_g[i,j] = \begin{cases} \Psi([x_i||x_j]), \text{ if}(i,j) \in \mathcal{E}, \\ 0, \text{otherwise}, \end{cases} \tag{3}$$

where $\boldsymbol{m}_g[i,j]$ denotes the edge score for e_{ij}, the graph masker Ψ takes the concatenation of embeddings of v_i and v_j as input and output the edge score. Practically, we employ a 2-layer MLP for its implementation. We then co-optimize $\boldsymbol{\Theta}$, and Ψ with the following loss function:

$$\mathcal{L}_{\text{anchor}} = \mathcal{L}\left(\{\boldsymbol{m}_g \odot \mathbf{A}, \mathbf{X}\}, \boldsymbol{\Theta}\right), \tag{4}$$

After the full graph training of E epochs (practically $E \leq 200$), we select the optimal mask $\boldsymbol{m}_g^{\mathcal{A}}$ and model output $\mathbf{Z}^{\mathcal{A}} = f(\{\boldsymbol{m}_g^{\mathcal{A}} \odot \mathbf{A}, \mathbf{X}\}, \boldsymbol{\Theta}^{\mathcal{A}})$ from the epoch with the highest validation score, collectively termed as the *anchor graph* $\mathcal{G}^{\mathcal{A}} = \{\boldsymbol{m}_g^{\mathcal{A}} \odot \mathbf{A}, \mathbf{Z}^{\mathcal{A}}\}$. This process is rooted in an intuitive concept: *although* $\boldsymbol{m}_g$ *and* $\boldsymbol{\Theta}$ *can be undertrained at this stage, the early training is capable of discovering vital connections and connectivity patterns* (Achille et al., 2018; You et al., 2019; 2022). Consequently, $\boldsymbol{m}_g^{\mathcal{A}}$ and $\mathbf{Z}^{\mathcal{A}}$ retain crucial properties for GNN training with the original graph $\mathcal{G}$. The anchor graph will be utilized in Sec. 3.5 to guide the exploration of sparse graph structure.

Given the target graph sparsity $s_g\%$, we zero the lowest-magnitude elements in $\boldsymbol{m}_g^{\mathcal{A}}$ w.r.t. s_g, and obtain its binarized version $\mathbf{M}_g^{\text{os}} \in \{0,1\}^{|\boldsymbol{m}_g^{\mathcal{A}}|}$. Such a sparse mask obtained via one-shot pruning is suboptimal (Ma et al., 2021; Frankle et al., 2020), and we will start from this point and continually refine towards a more optimal graph structure.

3.4. Dynamical Sparse Graph Training

With $\mathbf{M}_g^{\text{os}}$ at hand, we proceed to train the GNN model together with the fixed subgraph and the graph masker, denoted as $f(\{\boldsymbol{m}_g \odot \mathbf{M}_g^{\text{os}} \odot \mathbf{A}, \mathbf{X}\}, \boldsymbol{\Theta})$, with the objective function similar to Eq. 4. We aim to gradually evolve the sparse graph structure toward both better topological and semantical preservation. To this end, we periodically reactivate the semantically and topologically significant edges in the pruned subgraph and substitute them for less important portions in the current subgraph within D epochs.

We set the interval for each update (*i.e.*, drop $\leftrightarrow$ regrow) at ΔT epochs, with the total number of updates being $\lceil D/\Delta T \rceil$. We aim to develop a comprehensive criterion ϕ that evaluates the importance of an edge from both topo-

logical and semantic perspectives, which will guide the "exchange of edges" between the current edges $\mathcal{E}_{(\mu)} = \mathbf{M}_g \odot \mathbf{A}$ and its complement $\mathcal{E}^C_{(\mu)} = \neg\mathbf{M}_g \odot \mathbf{A}$. Consider the update process between interval μ and $\mu+1$:

$$\begin{aligned}\mathbf{M}^{(\text{prune})} &= \text{ArgTopK}\left(-\phi(\mathbf{M}_g^{(\mu)} \odot \mathbf{A}),\ \boldsymbol{\Upsilon}(\mu)\right),\\ \mathbf{M}^{(\text{regrow})} &= \text{ArgTopK}\left(\phi(\neg\mathbf{M}_g^{(\mu)} \odot \mathbf{A}),\ \boldsymbol{\Upsilon}(\mu)\right),\end{aligned} \tag{5}$$

where $\text{ArgTopK}(m,k)$ returns the indices of top-k elements of matrix m, and $\boldsymbol{\Upsilon}(\cdot)$ is the update scheduler that controls the number of edges to be swapped at each update. The configuration of $\boldsymbol{\Upsilon}(\cdot)$ is detailed in Appendix G.4. We then update the sparse graph as follows:

$$\mathbf{M}_g^{(\mu+1)} = \left(\mathbf{M}_g^{(\mu)} \setminus \mathbf{M}_g^{(\text{prune})}\right) \cup \mathbf{M}_g^{(\text{regrow})}. \tag{6}$$

Then, in $(\mu+1)$-th interval, we continue training the GNN with the updated sparse graph $\mathbf{M}_g^{(\mu+1)} \odot \mathbf{A}$ for another ΔT epochs. This iterative process of updating and refining the sparse graph structure aims towards optimal performance. The remaining question now is: *how do we design an ideal evaluation criterion ϕ*?

3.5. Topological & Semantical Criterion Design

To answer the question above, we introduce the Equilibria Principle to guide the design of graph pruning criteria, aiming to establish a new paradigm for sparsifier development:

Principle 3.1 (***Equilibria Sparsification Principle***). *Given a graph $\mathcal{G} = \{\mathbf{A}, \mathbf{X}\}$ and target sparsity $s_g\%$, an ideal sparsifier $\mathcal{M}_g \in \{0,1\}^{|\mathbf{A}|}(\frac{\|\mathcal{M}_g\|_0}{\|\mathcal{A}\|_0} = s_g\%)$ and its resulting subgraph $\mathcal{G}^{sub} = \{\mathcal{M}_g \odot \mathbf{A}, \mathbf{X}\}$ should satisfy the following condition:*

$$\arg\min_{\mathcal{M}_g} \left(\sum_{\mathcal{T}' \in \{\mathcal{T}\}} \mathcal{T}'(\mathcal{G}, \mathcal{G}^{sub}) + \sum_{\mathcal{S}' \in \{\mathcal{S}\}} \mathcal{S}'(\mathcal{G}, \mathcal{G}^{sub}) \right), \tag{7}$$

where $\{\mathcal{T}\}$ denotes all possible metrics measuring the topological information difference between the sparse and original graphs (graph distance, spectral gap, etc.), and $\{\mathcal{S}\}$ represents all possible metrics measuring semantic information differences (gradients, momentum, etc.).

However, it is impractical to traverse and satisfy all possible metrics, so we provide two exemplified approaches for topology/semantics preservation metrics $\mathcal{T}$ and $\mathcal{S}$, and utilize them to guide the update process in Sec. 3.4.

Topology Criterion. Despite the myriad of edge scoring methods (Tsitsulin & Perozzi, 2023), we opt for **eigenvalue variation**, *i.e.*, the relative error of all eigenvalues, as the metric for edge dropping/regrowing. This is because most topology-guided scores, including effective resistance (Spielman & Srivastava, 2008), spectral radius (Costa et al., 2007), and graph curvature (Tsitsulin & Perozzi, 2023; Forman, 2003), rely partially or wholly on graph Laplacian eigenvalues. Ideally, we aim to identify a set of edges $\mathcal{E}'$ from $\mathcal{E}_{(\mu)}$ that minimally impact the graph Laplacian and a set $\mathcal{E}''$ from $\mathcal{E}^C_{(\mu)}$ that maximally affect it ($|\mathcal{E}'| = |\mathcal{E}''| = \boldsymbol{\Upsilon}(\mu)$), thereby guiding the sparse graph structure towards restoring the topological properties of the anchor graph. The objective is as follows:

$$\arg\min_{\mathcal{E}',\mathcal{E}''} \mathbb{E}\left(\mathcal{T}(\mathcal{G}^{\mathcal{A}}, \mathcal{G}^{(\mu+1)})\right), \mathcal{T}(\mathcal{G}_1, \mathcal{G}_2) = \sum_{i=1}^{N} \frac{|\lambda_i^{(1)} - \lambda_i^{(2)}|}{\lambda_i^{(1)}}, \tag{8}$$

where $\mathcal{G}^{(\mu+1)} = \{\mathcal{V}, \mathcal{E}_{(\mu+1)}\}$ is the updated sparse graph after the μ-th update, and we choose eigenvalue variation as the implementation for $\mathcal{T}$. However, exhaustively evaluating all combinations of $(\mathcal{E}', \mathcal{E}'')$ is impractical. Therefore, we shift to assessing the impact of individual edges on the graph Laplacian as a measure of their importance:

$$\phi^{(\text{topo})}(e_{ij}) = \sum_{k=1}^{N} \frac{|\lambda_k(\mathcal{G}^{\mathcal{A}}) - \lambda_k(\mathcal{G}^{\mathcal{A}} \setminus e_{ij})|}{\lambda_k(\mathcal{G}^{\mathcal{A}})}, \tag{9}$$

where $\lambda_k(\mathcal{G}^{\mathcal{A}})$ denotes the k-th eigenvalue of the anchor graph, and $\lambda_k(\mathcal{G}^{\mathcal{A}} \setminus e_{ij})$ represents that after removing edge e_{ij}. For computational feasibility, we provide an approximate version, with detailed derivation in Appendix C:

$$\phi^{(\text{topo})}(e_{ij}) = \left(\sum_{k=1}^{K} + \sum_{k=N-K}^{N}\right) \frac{\boldsymbol{m}^{\mathcal{A}}_{g,ij} a_i^{(k)} b_j^{(k)}}{\lambda_k(\mathcal{G}^{\mathcal{A}}) a^{(k)T} b^{(k)}}, \tag{10}$$

where $a^{(k)}$ ($b^{(k)}$) is the left (right) eigenvector of the anchor graph's k-th eigenvalue. Notably, such topology criterion only needs to be computed once at the first update for subsequent reference.

Semantic Criterion. From the perspective of semantic preservation, we formulate the objective as follows:

$$\begin{aligned}&\arg\min_{\mathcal{E}',\mathcal{E}''} \mathbb{E}\left(\mathcal{S}(\mathcal{G}^{\mathcal{A}}, \mathcal{G}^{(\mu+1)})\right),\\ &\mathcal{S}(\mathcal{G}^{\mathcal{A}}, \mathcal{G}^{(\mu)}) = \mathbb{KL}\left(f(\mathcal{G}^{\mathcal{A}}, \boldsymbol{\Theta}^{\mathcal{A}}), f(\mathcal{G}^{(\mu)}, \boldsymbol{\Theta})\right),\end{aligned} \tag{11}$$

where $\mathbf{Z}^{\mathcal{A}} = f(\mathcal{G}^{\mathcal{A}}, \boldsymbol{\Theta}^{\mathcal{A}})$ is the semantic anchor, $f(\mathcal{G}^{(\mu)}, \boldsymbol{\Theta})$ is the current model output, and $\mathcal{S}(\cdot,\cdot)$ employ the KL divergence (Sun et al., 2022) to evaluate the model output discrepancies. Based on this, we propose the semantic criterion:

$$\phi^{(\text{sema})}(e_{ij}) = \left|\nabla_{e_{ij}} \mathcal{S}(\mathcal{G}^{\mathcal{A}}, \mathcal{G}^{(\mu)})\right|, \tag{12}$$

where edges with greater gradient magnitude after a single backpropagation step on output discrepancies are considered more valuable, otherwise the opposite.

Equilibria Combination. After selecting appropriate topological/semantic criteria, we combine these to form the final drop/regrow criterion, aligning with the equilibria principal:

$$\phi(e_{ij}) = \beta^s \cdot \phi^{(\text{sema})}(e_{ij}) + \beta^t \cdot \phi^{(\text{topo})}(e_{ij}), \tag{13}$$

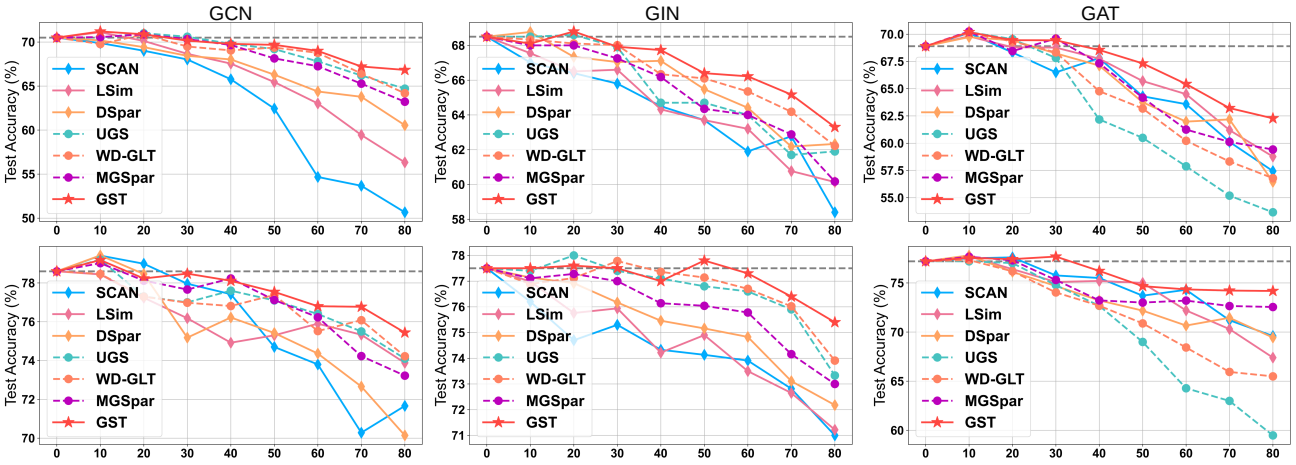

Figure 3. Performance comparison of graph sparsification methods on Citeseer (***First Row***) and PubMed (***Second Row***) under different sparsity levels. The gray dashed line represents the original baseline performance.

where β^s and β^t are corresponding scaling coefficients. At the end of interval μ, we exchange edges with the highest ϕ in $\mathcal{E}^C_{(\mu)}$ and those with the lowest ϕ in $\mathcal{E}^C_{(\mu)}$, resulting in the updated sparse graph $\mathcal{G}^{(\mu+1)}$, as described in Sec. 3.4.

Model Summary. Through such iterative exploration of graph structure, GST eventually achieves a sparse subgraph at the desired sparsity with no performance degradation. The overall algorithm framework is showcased in Algo. 1, and its complexity analysis is presented in Appendix D. Importantly, GST, as a versatile concept, exhibits compatibility with various mainstream research lines, thereby providing substantial support for advancements in these areas, such as graph adversarial defense and graph lottery ticket, *etc*. (discussed in Sec. 4.5).

4. Experiments

In this section, we conduct extensive experiments to answer the following research questions ($\mathcal{RQ}$):

- $\mathcal{RQ}$**1:** Can GST effectively find resilient sparse subgraphs?
- $\mathcal{RQ}$**2:** How effective is GST in terms of preserving spectral information, *i.e.*, eigenvalues?
- $\mathcal{RQ}$**3:** Does GST genuinely accelerate the GNN inference?
- $\mathcal{RQ}$**4:** Can GST serve as a universal operator?
- $\mathcal{RQ}$**5:** How sensitive is GST to its key hyperparameters and components?

4.1. Experiment Setup

Dataset. To exhaustively assess GST across various benchmarks and tasks, we select three popular graph datasets, including Cora, Citeseer, and PubMed (Kipf & Welling, 2017a). For larger-scale graphs, we utilize Ogbn-ArXiv, Ogbn-Proteins, and Ogbn-Products (Hu et al., 2020).

Backbones. We evaluate GST under both transductive and inductive settings. For **transductive settings**, we select GCN (Kipf & Welling, 2017b), GIN (Xu et al., 2019) and GAT (Veličković et al., 2018) for full-batch training. For **inductive settings**, we select GraphSAGE (Hamilton et al., 2017) and ClusterGCN (Chiang et al., 2019).

Baselines. We compare GST against two categories of graph sparsification methods: (1) **topology-based sparsification**, including SCAN (Xu et al., 2007), Local Similarity (LSim) (Satuluri et al., 2011), and DSpar (Liu et al., 2023); (2) **semantic-based sparsification**, including UGS (Chen et al., 2021), meta-gradient sparsifier (MGSpar) (Wan & Schweitzer, 2021), and WD-GLT (Hui et al., 2023). More details on experiment setup can be found in Appendix G.

4.2. GST Excels In Combating Sparsity ($\mathcal{RQ}$1)

We present the performance of GCN/GIN/GAT on Cora/Citeseer/PubMed in Figs. 3 and 8 and that of GraphSAGE/ClusterGCN on Ogbn-ArXiv/Proteins/Products in Tabs. 2, 9 and 10. Each point in the figures represents the test accuracy/ROCAUC of the GNNs with the corresponding sparsifier at various levels of graph sparsity. Our observations are as follows:

Obs.❶ GST is flexible and consistently outperforms other sparsifiers. Figs. 3 and 8 demonstrate that GST **(I)** maintains GNN performance best at lower sparsity levels, such as on GIN+PubMed with 50% graph sparsity, where other sparsifiers experienced a performance drop of $0.36\% \sim 3.17\%$ compared to the baseline, whereas GST even showed 0.3% improvement; **(II)** most effectively counters the adverse effects of sparsification at extreme sparsity levels, outperforming other sparsifiers by $1.23\% \sim 5.39\%$ at 80% graph sparsity.

Obs.❷ GST resiliently scales to large-scale graphs. As demonstrated in Tabs. 2, 9 and 10, GST maintains robust performance when sparsifying large graphs under inductive setting. Specifically, on Ogbn-ArXiv with 40% sparsity, GST+GraphSAGE/ClusterGCN experienced negligible per-

Table 2. Performance comparison of sparsifiers at different sparsity levels (10% → 60%) on GraphSAGE/ClusterGCN with Ogbn-ArXiv. We report the mean accuracy ± stdev of 3 runs. We shade the best-performing value in each column.

GraphSAGE	10%	20%	30%	40%	50%	60%
LSim	$69.22_{\pm 0.11}$	$68.40_{\pm 0.18}$	$66.15_{\pm 0.22}$	$64.66_{\pm 0.31}$	$61.07_{\pm 0.23}$	$58.21_{\pm 0.09}$
DSpar	$71.23_{\pm 0.24}$	$71.03_{\pm 0.28}$	$68.50_{\pm 0.33}$	$64.57_{\pm 0.26}$	$62.79_{\pm 0.66}$	$60.49_{\pm 0.58}$
UGS	$68.77_{\pm 0.21}$	$67.92_{\pm 0.53}$	$66.30_{\pm 0.27}$	$66.57_{\pm 0.18}$	$65.72_{\pm 0.14}$	$63.40_{\pm 0.36}$
GST	$71.12_{\pm 0.23}$	$71.14_{\pm 0.18}$	$71.46_{\pm 0.37}$	$70.57_{\pm 0.34}$	$68.02_{\pm 0.58}$	$66.55_{\pm 0.53}$
ClusterGCN	10%	20%	30%	40%	50%	60%
LSim	$67.27_{\pm 0.54}$	$67.80_{\pm 0.46}$	$65.49_{\pm 0.45}$	$64.97_{\pm 0.52}$	$63.56_{\pm 0.39}$	$62.18_{\pm 0.39}$
DSpar	$68.75_{\pm 0.75}$	$68.40_{\pm 0.69}$	$66.72_{\pm 0.56}$	$66.09_{\pm 0.71}$	$65.45_{\pm 0.51}$	$63.72_{\pm 0.50}$
GST	$69.44_{\pm 0.14}$	$69.21_{\pm 0.24}$	$68.17_{\pm 0.63}$	$68.02_{\pm 0.49}$	$67.33_{\pm 0.37}$	$65.98_{\pm 0.50}$

formance losses of only 0.43% and 0.48%, respectively. For Ogbn-Products+ClusterGCN at 60% sparsity, GST outperforms DSpar and LSim by 3.45% and 7.95%, respectively.

Obs.❸ Different GNN backbones and graphs show varying resistance to sparsification. As shown in Fig. 3, GIN/GAT are less affected by graph sparsification compared to GCN. At 80% graph sparsity, sparsifiers generally lead to an accuracy drop of 7.08% ∼ 20.7% on GCN+Cora, whereas that on GAT+Cora is only 6.44% ∼ 14.1%. Moreover, the resilience to sparsification varies across graphs; specifically, GAT+Citeseer experiences a 3.67% ∼ 19.9% decline at 80% sparsity, while GAT+PubMed shows only a 3.16% ∼ 8.46% degradation.

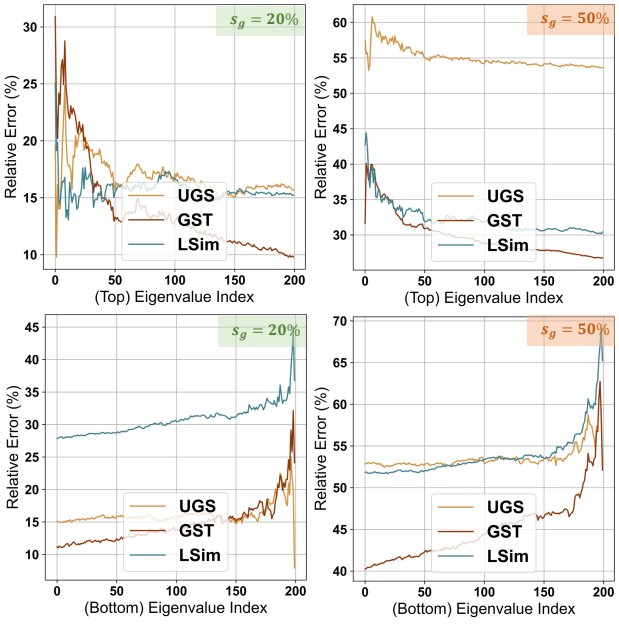

Figure 4. The relative error of the top-200 and bottom-200 eigenvalues on PubMed+GCN, *i.e.*, $\frac{\lambda_i - \lambda_i'}{\lambda_i}$, sparsified by different methods at sparsity level 20% and 50%.

4.3. Spectral Preservation Helps Sparsification ($\mathcal{RQ}2$)

To answer $\mathcal{RQ}2$, we compare the eigenvalue variation between sparse and original graphs, following (Liu et al., 2023). In Fig. 4, we showcase the relative error in the top-200 and bottom-200 eigenvalues of sparse graphs produced by sparsifiers compared to the original graph, on Citeseer/PubMed+GIN/GAT. We select only the top/bottom-200 eigenvalues because small (large) eigenvalues represent the global clustering (local smoothness) structures of the graphs, adequately reflecting the preservation of spectral information (Liu et al., 2023). We observe that:

Obs.❹ GST effectively preserves key eigenvalues. As demonstrated in Figs. 4 and 8 to 10, for the top-200 eigenvalues, GST achieves a performance similar to or better than LSim, and significantly surpasses UGS at 50% sparsity. On Citeseer+GIN, its relative error fluctuates only within 0% ∼ 8%. Regarding the bottom-200 eigenvalues, GST provides the best approximation. At 20% sparsity, its average relative error is 20% lower than that of UGS. More analyses can be found in Appendix H.2.

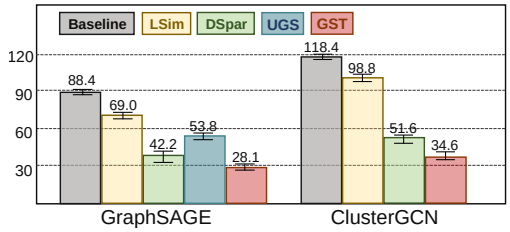

Figure 5. The inference latency on Ogbn-Proteins with different sparsifiers when their performance loss is negligible ($\leq 1\%$).

4.4. GST Significantly Accelerates Computations ($\mathcal{RQ}3$)

To answer $\mathcal{RQ}3$, we exhaustively compare the efficiency of different sparsifiers via three metrics, extreme graph sparsity, inference MACs, and GPU inference latency in Tab. 3 and Figs. 5 and 11. Further explanations on these metrics can be found in Appendix G.6. We give the following observations:

Obs.❺ GST significantly accelerates GNN inference. More specifically, GST's inference speedup is more pronounced for large-scale graphs compared to smaller ones. As shown in Tab. 3, GST provides 33.29% ∼ 37.22% MACs savings for GCN/GIN/GAT, with average inference speedups of 1.30×, 1.37×, and 1.45×, respectively. On Ogbn-Proteins, this can reach 3.14 ∼ 3.42× (in Fig. 5). In conclusion, GST significantly aids in accelerating GNN inference at a practically applicable level.

4.5. High Robustness and Versatility of GST ($\mathcal{RQ}4$)

To validate the versatility of GST, this section examines (1) its robustness against edge perturbations, and (2) its applicability to graph lottery tickets identification. In the perturbation experiments, we induce edge perturbations by randomly relocating edge endpoints. For the graph lottery ticket experiment, we simply replace the graph mask found by UGS (Chen et al., 2021) with that by GST in the graph lottery ticket and retrain the GNN from scratch. Figs. 6 and 12 illustrate the performance of GST at various spar-

Table 3. Efficiency comparison between GST and other sparsifiers. "Sparsity (%)" indicates the extreme graph sparsity of different sparsifiers; "MACs (m)" represents the inference MACs (= $\frac{1}{2}$FLOPs); "Latency (ms)" refers to GPU reference latency. We shade the best results and underline the second-best results for each dataset. The GPU latency results are averaged over five random trials.

	Model	Cora			Citeseer			PubMed			Avg.		
		Sparsity (%)	MACs (m)	Latency (ms)	Sparsity (%)	MACs (m)	Latency (ms)	Sparsity (%)	MACs (m)	Latency (ms)	Rank	MAC Savings	Speedup
GCN	Baseline	–	1996.55	4.27	–	6318.00	6.08	–	5077.84	6.25	–	0%	1.00×
	SCAN	11.93%	1758.37	4.13	6.52%	5526.99	5.88	23.68%	3875.41	5.52	**7**	16.04%	1.03×
	LSim	11.58%	1765.35	3.99	21.83%	4938.78	5.35	11.84%	4476.62	5.67	**5**	15.06%	1.10×
	DSpar	9.78%	1882.44	4.20	8.22%	5943.22	5.97	18.60%	4266.18	5.49	**6**	8.65%	1.05×
	UGS	19.10%	1758.37	4.13	33.80%	4166.50	4.97	24.91%	3905.67	5.12	**3**	23.73%	1.13×
	WD-GLT	19.46%	1702.34	4.24	22.70%	4873.16	5.33	11.24%	4455.10	5.67	**4**	18.20%	1.11×
	MGSpar	31.00%	1568.21	3.92(1.08×)	30.00%	4318.75	5.03(1.20×)	27.00%	3758.53	5.11(1.22×)	**2**	24.67%	1.13×
	GST	40.00%	1397.59	3.36(1.27×)	43.00%	3890.80	4.48(1.35×)	35.00%	3092.81	4.82(1.29×)	**1**	34.49%	1.30×
GIN	Baseline	–	2006.26	2.53	–	6328.22	5.02	–	5108.12	6.12	–	0%	1.00×
	SCAN	11.93%	1766.91	2.24	12.52%	5535.93	3.84	11.82%	4504.34	5.81	**5**	12.10%	1.16×
	LSim	11.58%	1773.93	2.34	10.43%	5668.19	4.11	11.84%	4503.32	5.74	**6**	11.28%	1.12×
	DSpar	21.11%	1533.29	2.02(1.25×)	14.77%	5372.09	3.71	16.18%	4157.10	5.16	**3**	19.09%	1.26×
	UGS	19.4%	1617.05	2.18	27.5%	4587.96	3.60(1.39×)	27.6%	3698.28	5.02	**4**	24.83%	1.25×
	WD-GLT	19.7%	1590.33	2.16	16.4%	5198.11	3.73	40.22%	3318.59	4.75(1.25×)	**2**	23.66%	1.27×
	MGSpar	10.00%	1784.52	2.39	5.00%	6087.18	4.90	7.00%	4885.16	5.94	**7**	6.49%	1.05×
	GST	31.00%	1304.27	1.85(1.36×)	26.43%	4633.97	3.52(1.42×)	53.00%	3109.47	4.57(1.33×)	**1**	33.29%	1.37×
GAT	Baseline	–	8029.60	13.2	–	25309.91	13.5	–	20675.00	15.3	–	0%	1.00×
	SCAN	24.16%	6089.65	9.9	24.87%	19015.33	11.8	23.68%	15779.16	12.6(1.21×)	**3**	22.23%	1.23×
	LSim	23.22%	6165.13	10.2	21.83%	19784.75	11.8	11.84%	18227.08	14.9	**6**	16.96%	1.15×
	DSpar	17.34%	6788.29	11.33	22.85%	19655.05	11.7	11.58%	18106.99	14.8	**7**	15.25%	1.12×
	UGS	18.22%	6584.27	11.15	25.54%	18850.75	12.36	18.22%	16953.50	13.92	**6**	18.66×	1.13×
	WD-GLT	19.71%	6504.33	10.91(1.20×)	28.54%	18223.13	11.57	13.54%	17784.61	14.50	**4**	18.59%	1.16×
	MGSpar	19.00%	6400.25	11.08	32.00%	17114.76	11.19(1.20×)	20.00%	16309.54	12.8	**2**	27.59%	1.23×
	GST	35.00%	5216.28	7.90(1.67×)	41.00%	14920.45	10.30(1.31×)	39.00%	13438.75	11.23(1.36×)	**1**	37.22%	1.45×

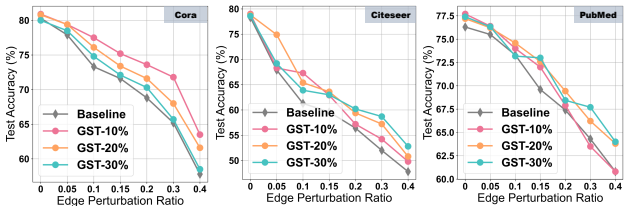

Figure 6. The robust performance of GST on edge perturbations with a varying fraction of perturbed edges ($0\% \rightarrow 40\%$).

sity levels against edge perturbation, and Tab. 11 presents the performance improvement achieved by combining GST with UGS. Observations include: ❶ GST significantly enhances GNN's robustness against edge perturbations. For instance, on Cora+GCN, GST at 10% sparsity achieved up to 7.23% performance improvement (in Fig. 12). ❷ Across all sparsity levels, GST consistently aids UGS in identifying graph lottery tickets (in Tab. 11). Detailed analysis is provided in Appendix H.4.

4.6. Ablation & Parameter Sensitivity Analysis ($\mathcal{RQ}5$)

Ablation Study. In order to better verify the effectiveness of each component of GST, We have made corresponding modifications to GST and designed the following three variants: (1) **GST *w/o* tuning**: moving the dynamic sparse training part of GST; (2) **GST *w/o* sema**: merely utilizing $\phi^{(\text{sema})}$ when updating the sparse graph; (3) **GST *w/o* topo**: merely utilizing $\phi^{(\text{topo})}$ when updating the sparse graph.

As indicated in Tab. 4 and Fig. 13, it is evident that ❶ the removal of the dynamic fine-tuning process in GST *w/o* tuning leads to a significant performance drop; ❷ using either $\phi^{(\text{topo})}$ or $\phi^{(\text{sema})}$ alone cannot match the performance of the original GST, with the omission of $\phi^{(\text{sema})}$ having a more pronounced impact. In summary, removing either component deteriorates the effectiveness of GST. For detailed data, as well as parameter sensitivity experiments, refer to Appendix H.5.

Table 4. Ablation study on GST with its three variants. We report the extreme graph sparsity on Citeseer+GCN/GIN and Ogbn-ArXiv+GraphSAGE/ClusterGCN.

Dataset	Citeseer		Ogbn-ArXiv	
Backbone	GCN	GIN	GraphSAGE	ClusterGCN
GST	$43.43_{\pm 1.46}$	$26.43_{\pm 1.07}$	$35.18_{\pm 0.72}$	$21.44_{\pm 0.95}$
GST *w/o* tuning	$30.41_{\pm 0.75}$	$10.39_{\pm 1.78}$	$18.32_{\pm 1.14}$	$12.57_{\pm 0.56}$
GST *w/o* sema	$39.82_{\pm 1.49}$	$25.09_{\pm 1.32}$	$29.78_{\pm 1.42}$	$19.24_{\pm 0.85}$
GST *w/o* topo	$42.08_{\pm 0.56}$	$25.93_{\pm 1.26}$	$32.21_{\pm 0.49}$	$19.67_{\pm 0.79}$

5. Conclusion & Future Work

This paper studies the notorious inefficiency of GNNs. Different from previous research literature, we open a novel topic, termed Graph Sparse Training (GST), for the first time. GST aims to fine-tune the sparse graph structure during the training process, utilizing anchors derived from full graph training as supervisory signals. GST also proposes the Equilibria Principle to balance both topological and semantic information preservation. Extensive experiments demonstrate that integrating the GST concept can enhance both performance and inference speedup. Additionally, GST can serve as a philosophy to benefit a wide array of training algorithms. In the future, we plan to extend GST to more complex scenarios, including heterophilic and heterogeneous graphs, and further explore the feasibility of in-time sparsification & acceleration in real-world applications (recommender systems, fraud detection, *etc.*).

Acknowledgement

Yuxuan Liang is in part supported by Guangzhou-HKUST (GZ) Joint Funding Program (No. 2024A03J0620) and Guibin Zhang is in part supported by the Tongji University Undergraduate Innovation and Entrepreneurship Program (No. 202310247097).

Impact Statement

Ethical impacts. We confidently affirm that our paper is free of ethical concerns, encompassing its motivation, design, experiments, and the data utilized. The proposed GST is designed to advance the field of graph sparsification and graph neural network acceleration, ensuring that our research contributes positively and responsibly to the scientific community.

Expected societal implications. Training GNNs on actual graph data presents a substantial computational burden. GST offers a more efficient computational approach by reducing the edges within the graph data. This downsizing significantly lowers the energy consumption of computing devices, thereby reducing carbon emissions. Such a decrease in carbon emissions greatly benefits the environment and aligns with ongoing sustainability efforts. Furthermore, with the growing ubiquity of large (graph) models, GST provides a low-cost solution, offering efficient insights for future model deployment. This dual benefit not only advances the technical efficiency of graph-based computations but also promotes environmental sustainability and economic feasibility in training complex models.

References

Achille, A., Rovere, M., and Soatto, S. Critical learning periods in deep networks. In *International Conference on Learning Representations*, 2018.

Batson, J., Spielman, D. A., Srivastava, N., and Teng, S.-H. Spectral sparsification of graphs: theory and algorithms. *Communications of the ACM*, 56(8):87–94, 2013.

Benczúr, A. A. and Karger, D. R. Approximating st minimum cuts in õ (n 2) time. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pp. 47–55, 1996.

Bhattacharjee, R., Dexter, G., Drineas, P., Musco, C., and Ray, A. Sublinear time eigenvalue approximation via random sampling. *arXiv preprint arXiv:2109.07647*, 2021.

Chan, H. and Akoglu, L. Optimizing network robustness by edge rewiring: a general framework. *Data Mining and Knowledge Discovery*, 30:1395–1425, 2016.

Chen, J., Zhu, J., and Song, L. Stochastic training of graph convolutional networks with variance reduction. *arXiv preprint arXiv:1710.10568*, 2017.

Chen, J., Ma, T., and Xiao, C. Fastgcn: fast learning with graph convolutional networks via importance sampling. In *Proceedings of ICLR*, 2018.

Chen, T., Sui, Y., Chen, X., Zhang, A., and Wang, Z. A unified lottery ticket hypothesis for graph neural networks. In *International Conference on Machine Learning*, pp. 1695–1706. PMLR, 2021.

Chen, Y., Ye, H., Vedula, S., Bronstein, A., Dreslinski, R., Mudge, T., and Talati, N. Demystifying graph sparsification algorithms in graph properties preservation. *arXiv preprint arXiv:2311.12314*, 2023.

Chiang, W.-L., Liu, X., Si, S., Li, Y., Bengio, S., and Hsieh, C.-J. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 257–266, 2019. doi: 10.1145/3292500.3330925.

Costa, L. d. F., Rodrigues, F. A., Travieso, G., and Villas Boas, P. R. Characterization of complex networks: A survey of measurements. *Advances in physics*, 56(1): 167–242, 2007.

Cui, G. and Wei, Z. Mgnn: Graph neural networks inspired by distance geometry problem. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 335–347, 2023.

David, J. Algorithms for analysis and design of robust controllers. 1995.

Dettmers, T., Zettlemoyer, L., and Zhang. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019.

Eden, T., Jain, S., Pinar, A., Ron, D., and Seshadhri, C. Provable and practical approximations for the degree distribution using sublinear graph samples. In *Proceedings of the 2018 World Wide Web Conference*, pp. 449–458, 2018.

Evci, U., Gale, T., Menick, J., Castro, P. S., and Elsen, E. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, pp. 2943–2952. PMLR, 2020.

Fang, J., Li, X., Sui, Y., Gao, Y., Zhang, G., Wang, K., Wang, X., and He, X. Exgc: Bridging efficiency and explainability in graph condensation. *arXiv preprint arXiv:2402.05962*, 2024.

Fey, M. and Lenssen, J. E. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

Forman. Bochner's method for cell complexes and combinatorial ricci curvature. *Discrete & Computational Geometry*, 29:323–374, 2003.

Frankle, J., Dziugaite, G. K., Roy, D. M., and Carbin, M. Pruning neural networks at initialization: Why are we missing the mark? *arXiv preprint arXiv:2009.08576*, 2020.

Hamann, M., Lindner, G., Meyerhenke, H., Staudt, C. L., and Wagner, D. Structure-preserving sparsification methods for social networks. *Social Network Analysis and Mining*, 6:1–22, 2016.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *Proceedings of NIPS*, 2017.

Hoang, D., Liu, S., Marculescu, R., and Wang, Z. Revisiting pruning at initialization through the lens of ramanujan graph. In *International Conference on Learning Representations*. International Conference on Learning Representations (ICLR) 2023, 2023.

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. *arXiv preprint arXiv:2005.00687*, 2020.

Huang, S., Lei, B., Xu, D., Peng, H., Sun, Y., Xie, M., and Ding, C. Dynamic sparse training via balancing the exploration-exploitation trade-off. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6. IEEE, 2023.

Hui, B., Yan, D., Ma, X., and Ku, W.-S. Rethinking graph lottery tickets: Graph sparsity matters. In *The Eleventh International Conference on Learning Representations*, 2023.

Jin, W., Zhao, L., Zhang, S., Liu, Y., Tang, J., and Shah, N. Graph condensation for graph neural networks. *arXiv preprint arXiv:2110.07580*, 2021.

Karhadkar, K., Banerjee, P. K., and Montufar, G. Fosr: First-order spectral rewiring for addressing oversquashing in gnns. In *The Eleventh International Conference on Learning Representations*, 2023.

Kipf, T. N. and Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the 5th International Conference on Learning Representations*, 2017a.

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

Lee, N., Ajanthan, T., and Torr, P. H. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

Liu, J., Xu, Z., Shi, R., Cheung, R. C., and So, H. K. Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers. *arXiv preprint arXiv:2005.06870*, 2020.

Liu, S., Yin, L., Mocanu, D. C., and Pechenizkiy, M. Do we actually need dense over-parameterization? in-time over-parameterization in sparse training. In *International Conference on Machine Learning*, pp. 6989–7000. PMLR, 2021.

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

Luo, D., Cheng, W., Yu, W., Zong, B., Ni, J., Chen, H., and Zhang, X. Learning to drop: Robust graph neural network via topological denoising. In *Proceedings of the 14th ACM international conference on web search and data mining*, pp. 779–787, 2021.

Ma, X., Yuan, G., Shen, X., Chen, T., Chen, X., Chen, X., Liu, N., Qin, M., Liu, S., Wang, Z., et al. Sanity checks for lottery tickets: Does your winning ticket really win the jackpot? *Advances in Neural Information Processing Systems*, 34:12749–12760, 2021.

Mocanu, D. C., Mocanu, E., Stone, P., Nguyen, P. H., Gibescu, M., and Liotta, A. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9 (1):2383, 2018.

Qiu, S., You, L., and Wang, Z. Optimizing sparse matrix multiplications for graph neural networks. In *International Workshop on Languages and Compilers for Parallel Computing*, pp. 101–117. Springer, 2021.

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

Rozemberczki, B., Allen, C., and Sarkar, R. Multi-scale attributed node embedding. *Journal of Complex Networks*, 9(2):cnab014, 2021.

Satuluri, V., Parthasarathy, S., and Ruan, Y. Local graph sparsification for scalable clustering. In *Proceedings of*

the 2011 ACM SIGMOD International Conference on Management of data, pp. 721–732, 2011.

Spielman, D. A. and Srivastava, N. Graph sparsification by effective resistances. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pp. 563–568, 2008.

Sun, Q., Li, J., Peng, H., Wu, J., Fu, X., Ji, C., and Philip, S. Y. Graph structure learning with variational information bottleneck. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 4165–4174, 2022.

Tsitsulin, A. and Perozzi, B. The graph lottery ticket hypothesis: Finding sparse, informative graph structure. *arXiv preprint arXiv:2312.04762*, 2023.

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. *stat*, 1050: 20, 2017.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Wan, G. and Kokel, H. Graph sparsification via meta-learning. *DLG@ AAAI*, 2021.

Wan, G. and Schweitzer, H. Edge sparsification for graphs via meta-learning. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pp. 2733–2738. IEEE, 2021.

Wang, K., Liang, Y., Wang, P., Wang, X., Gu, P., Fang, J., and Wang, Y. Searching lottery tickets in graph neural networks: A dual perspective. In *The Eleventh International Conference on Learning Representations*, 2022.

Wang, K., Liang, Y., Li, X., Li, G., Ghanem, B., Zimmermann, R., Yi, H., Zhang, Y., Wang, Y., et al. Brave the wind and the waves: Discovering robust and generalizable graph lottery tickets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023a.

Wang, K., Liang, Y., Wang, P., Wang, X., Gu, P., Fang, J., and Wang, Y. Searching lottery tickets in graph neural networks: A dual perspective. In *The Eleventh International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=Dvs-a3aymPe.

Wang, M., Zheng, D., Ye, Z., Gan, Q., Li, M., Song, X., Zhou, J., Ma, C., Yu, L., Gai, Y., et al. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

Wang, Y., Liu, S., Chen, K., Zhu, T., Qiao, J., Shi, M., Wan, Y., and Song, M. Adversarial erasing with pruned elements: Towards better graph lottery ticket. *arXiv preprint arXiv:2308.02916*, 2023c.

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? In *International Conference on Learning Representations*, 2019.

Xu, X., Yuruk, N., Feng, Z., and Schweiger, T. A. Scan: a structural clustering algorithm for networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 824–833, 2007.

Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., and Leskovec, J. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of KDD*, 2018a.

Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., and Leskovec, J. Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems*, 31, 2018b.

You, H., Li, C., Xu, P., Fu, Y., Wang, Y., Chen, X., Baraniuk, R. G., Wang, Z., and Lin, Y. Drawing early-bird tickets: Towards more efficient training of deep networks. *arXiv preprint arXiv:1909.11957*, 2019.

You, H., Lu, Z., Zhou, Z., Fu, Y., and Lin, Y. Early-bird gcns: Graph-network co-optimization towards more efficient gcn training and inference via drawing early-bird lottery tickets. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 8910–8918, 2022.

Yuan, G., Ma, X., Niu, W., Li, Z., Kong, Z., Liu, N., Gong, Y., Zhan, Z., He, C., Jin, Q., et al. Mest: Accurate and fast memory-economic sparse training framework on the edge. *Advances in Neural Information Processing Systems*, 34: 20838–20850, 2021.

Zhang, G., Wang, K., Huang, W., Yue, Y., Wang, Y., Zimmermann, R., Zhou, A., Cheng, D., Zeng*, J., and Liang*, Y. Graph lottery ticket automated. In *The International Conference on Learning Representations*, 2024.

Zhang, J., Dong, Y., Wang, Y., Tang, J., and Ding, M. Prone: Fast and scalable network representation learning. In *IJCAI*, volume 19, pp. 4278–4284, 2019.

Zhang, M. and Chen, Y. Link prediction based on graph neural networks. In *Proceedings of NIPS*, 2018.

Zhang, M. and Chen, Y. Inductive matrix completion based on graph neural networks. *arXiv preprint arXiv:1904.12058*, 2019.

Zhang, Y., Zhao, L., Lin, M., Sun, Y., Yao, Y., Han, X., Tanner, J., Liu, S., and Ji, R. Dynamic sparse no training: Training-free fine-tuning for sparse llms. *arXiv preprint arXiv:2310.08915*, 2023.

Zheng, C., Zong, B., Cheng, W., Song, D., Ni, J., Yu, W., Chen, H., and Wang, W. Robust graph representation learning via neural sparsification. In *International Conference on Machine Learning*, pp. 11458–11468. PMLR, 2020.

Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., and Sun, M. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.

Zhu, M. and Gupta, S. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

A. Notations

Table 5. The notations that are commonly used in Methodology (Sec. 3).

Notation	Definition
$\mathcal{G} = \{\mathcal{V}, \mathcal{E}\} = \{\mathbf{A}, \mathbf{X}\}$	Input graph
$\mathbf{A}$	Input adjacency matrix
$\mathbf{X}$	Node features
$\mathbf{\Theta}$	Weight matrices of GNN
$\lambda_i(\mathcal{G})$	The i-th eigenvalue of $\mathcal{G}$
s_g	Target graph sparsity
Ψ	Graph masker
$\boldsymbol{m}_g^{\mathcal{A}}$	Graph mask at the point of the highest validation score during full-graph training
$\mathbf{Z}^{\mathcal{A}} = f(\{\boldsymbol{m}_g^{\mathcal{A}} \odot \mathbf{A}, \mathbf{X}\}, \mathbf{\Theta}^{\mathcal{A}})$	GNN output at the point of the highest validation score during full-graph training
$\mathcal{G}^{\mathcal{A}} = \{\boldsymbol{m}_g^{\mathcal{A}} \odot \mathbf{A}, \mathbf{Z}^{\mathcal{A}}\}$	Anchor graph that contains the topological & semantic information of the original graph
$\mathbf{M}_g^{\text{OS}}$	(Binary) one-shot graph mask
$\mathbf{M}_g^{(\mu)}$	(Binary) graph mask at μ-th interval
$\mathcal{E}_{(\mu)}$	Remained edges at μ-th interval
$\mathcal{E}_{(\mu)}^{C}$	Pruned edges at μ-th interval

B. More Dicussions on Related Work

Sprase Training	Target	Semantic?	Topology?	PRC§	Criterion†	Backbone
SNIP (Lee et al., 2018)	Sparse NN	✓	✗	✓	magnitude, gradient	AlexNet, VGG, LSTM, *etc.*
SET (Mocanu et al., 2018)	Sparse NN	✓	✗	✓	magnitude, gradient	MLP, CNN
SNFS (Dettmers et al., 2019)	Sparse NN	✓	✗	✓	magnitude, momentum	AlexNet, VGG, ResNet, *etc.*
RigL (Evci et al., 2020)	Sparse NN	✓	✗	✓	magnitude, gradient	ResNet, MobileNet
ITOP (Liu et al., 2021)	Sparse NN	✓	✗	✓	magnitude, gradient	MLP, VGG, ResNet, *etc.*
DST (Liu et al., 2020)	Sparse NN	✓	✗	✗	magnitude	LeNet, LSTM, VGG, *etc.*
MEST (Yuan et al., 2021)	Sparse NN	✓	✗	✓	magnitude, gradient	ResNet
IMDB (Hoang et al., 2023)	Sparse NN	✗	✓	✓	IMDB property	ResNet, VGG
DSnT (Zhang et al., 2023)	Sparse LLM	✓	✗	✓	magnitude, output variation	LLaMA, Vicuna, OPT
GST (Ours)	Sparse Graph	✓	✓	✓	magnitude, eigenvalue, gradient	GCN, GIN, GAT, *etc.*

§ Prune Rate Control. Whether the method has control over the sparsity rate.
† Criterion refers to the drop or regrow criterion during connection update.

Table 6. Comparison among different sparse training techniques, regarding their sparsification target, semantic and topological awareness, sparsity rate controllability, drop/grow criterion and backbones. Note that our proposed GST is the initial endeavor to apply the concept of dynamic sparse training to graph data, incorporating graph-related domain knowledge.

To highlight our contributions, we detail in Tab. 6 how our proposed GST differs from traditional (dynamic) sparse training methods, with the two most crucial distinctions as follows:

- **Sparsification Target**: Previous dynamic sparse training methods primarily focused on pruning parameters, typically within traditional CNN frameworks (*e.g.*, VGG, Resnet) or large language models (*e.g.*, LLaMA, OPT). In contrast, GST is a novel exploration of dynamically sparsifying the input data, specifically graph data.

- **Drop & Regrow Criterion**: Most prior sparse training techniques utilize semantic information from the model training procedure (*e.g.*, magnitude, momentum, gradients) for parameter pruning. GST is not a trivial adaptation of DST to graph data; instead, it represents the first attempt that combines well-developed semantic criteria and domain knowledge about graph topology. Specifically, GST, for the first time, considers both the impact of sparsification on model output (reflecting semantics) and on graph Laplacian eigenvalues (reflecting topology). It optimizes these two objectives in tandem to obtain the optimal sparse subgraph.

C. Eigenvalue Variation Approximation

Given the anchor graph $\mathcal{G}^{\mathcal{A}} = \{\boldsymbol{m}^{\mathcal{A}} \odot \mathbf{A}, \mathbf{Z}^{\mathcal{A}}\}$, we aim to assess how removing a specific edge affects $\mathcal{G}^{\mathcal{A}}$'s overall topological properties, as indicated by the eigenvalue variation, as follows:

$$\phi^{(\text{topo})}(e_{ij}) = \sum_{k=1}^{N} \frac{|\Delta\lambda_k|}{\lambda_k} = \sum_{k=1}^{N} \frac{|\lambda_k(\mathcal{G}^{\mathcal{A}}) - \lambda_k(\mathcal{G}^{\mathcal{A}} \setminus e_{ij})|}{\lambda_k(\mathcal{G}^{\mathcal{A}})}, \tag{14}$$

However, recalculating the N eigenvalues for each $\mathcal{G}^{\mathcal{A}} \setminus e_{ij}$ theoretically requires $\mathcal{O}(E \cdot N^3)$ complexity, which is computationally impractical. Therefore, we consider approximating the eigenvalue variation.

We denote the impact of removing e_{ij} on the anchor graph as $\Delta\mathbf{A}$, where $\Delta\mathbf{A}_{ij} = -1$, and the rest are zeros. The anchor graph after removing e_{ij} can be expressed as $\boldsymbol{m}_g^{\mathcal{A}} \odot (\mathbf{A} + \Delta\mathbf{A})$. Similarly, the influence of removing e_{ij} on the right eigenvalue λ_k and the corresponding right eigenvector $b^{(k)}$ of $\mathcal{G}^{\mathcal{A}}$ is denoted as $\Delta\lambda_k$ and Δb, respectively. According to the definitions of eigenvalues and eigenvectors, we have:

$$\left(\boldsymbol{m}_g^{\mathcal{A}} \odot (\mathbf{A} + \Delta\mathbf{A})\right)(b^{(k)} + \Delta b^{(k)}) = (\lambda_k + \Delta\lambda_k)(b^{(k)} + \Delta b^{(k)}). \tag{15}$$

It is noteworthy that for large matrices, it is reasonable to assume that the removal of a link or node has a minor impact on the spectral properties of the graph. Therefore, both $\Delta b^{(k)}$ and $\Delta\lambda^{(k)}$ are small. Left-multiplying this equation by the transpose of the left eigenvector a^T and neglecting second-order terms $a^{(k)T}\Delta\mathbf{A}\Delta b^{(k)}$ and $\Delta\lambda_k a^{(k)T}\Delta b^{(k)}$, we have:

$$\Delta\lambda_k = \frac{\boldsymbol{m}_{g,ij}^{\mathcal{A}} a_i^{(k)} b_j^{(k)}}{a^{(k)T} b^{(k)}}, \tag{16}$$

Based on this, the eigenvalue variation of e_{ij} can be further expressed as:

$$\phi^{(\text{topo})}(e_{ij}) = \sum_{k=1}^{N} \left| \frac{\boldsymbol{m}_{g,ij}^{\mathcal{A}} a_i^{(k)} b_j^{(k)}}{\lambda_k a^{(k)T} b^{(k)}} \right|, \tag{17}$$

However, for large graphs, computing all eigenvalues/eigenvectors still incurs significant computational overhead. Considering that small (large) eigenvalue can effectively indicate the global clustering (local smoothness) structure of the graphs (Zhang et al., 2019), we only select the top-K and bottom-K (K=20) eigenvalues to compute their variation:

$$\phi^{(\text{topo})}(e_{ij}) = \left(\sum_{k=1}^{K} + \sum_{k=N-K}^{N}\right) \left| \frac{\boldsymbol{m}_{g,ij}^{\mathcal{A}} a_i^{(k)} b_j^{(k)}}{\lambda_k a^{(k)T} b^{(k)}} \right|, \tag{18}$$

Additionally, we utilize non-trivial approximations of eigenvalues/eigenvectors with sublinear, *i.e.*, $o(N^2)$, time complexity (Bhattacharjee et al., 2021) to reduce the computational budget further.

D. Compelxity Analysis

During the full-graph training stage, the inference time complexity of GST is:

$$\mathcal{O}\left(\underbrace{L \times E \times D}_{\text{aggregation}} + \overbrace{E \times D}^{\text{graph masker}} + \underbrace{L \times N \times D}_{\text{update}}\right), \tag{19}$$

where L is the number of GNN layers and D is the feature dimension. Subsequently, we approximate the eigenvalue/eigenvector for each edge, with a sublinear time complexity of $o(N^2)$. It is worth noting that this computation is performed only once. The inference time complexity of the sparse training procedure is:

$$\mathcal{O}\left(\underbrace{L \times \|\mathbf{M}_g \odot \mathbf{A}\|_0 \times D}_{\text{sparsified aggregation}} + \overbrace{\|\mathbf{M}_g \odot \mathbf{A}\|_0 \times D}^{\text{graph masker}} + \underbrace{L \times N \times D}_{\text{update}}\right), \tag{20}$$

Algorithm 1 Algorithm workflow of **GST**

Input : $\mathcal{G} = (\mathbf{A}, \mathbf{X})$, GNN model $f(\mathcal{G}, \mathbf{\Theta}_0)$, GNN's initialization $\mathbf{\Theta}_0$, target sparsity $s_g\%$, update interval ΔT, the number of epochs to obtain anchor E, the number of epochs for sparse graph fine-tuning D, learning rate η.

Output : Sparse graph $\mathcal{G}^{\text{sub}} = \{\mathbf{M}_g \odot \mathbf{A}, \mathbf{X}\}$

for iteration $i \leftarrow 1$ **to** E **do**
  Compute the edge mask $\boldsymbol{m}_g^i$ via graph masker Ψ ; ▷ Eq. 3
  Forward $f_{sub}\left(\{\boldsymbol{m}_g^i \odot \mathbf{A}, \mathbf{\Theta}_i\}, \boldsymbol{m}_\theta\right)$ to compute $\mathcal{L}_{\text{anchor}}$; ▷ Eq. 4
  Backpropagate to update $\mathbf{\Theta}_{i+1} \leftarrow \mathbf{\Theta}_i - \eta \nabla_{\mathbf{\Theta}} \mathcal{L}_{\text{anchor}}$.
  Update masks $\boldsymbol{m}_g$ with graph masker Ψ.
end
/* Obtain Anchor Graph */
Record the anchor graph $\mathcal{G}^{\mathcal{A}} = \{m_g^{\mathcal{A}} \odot \mathbf{A}, \mathbf{Z}^{\mathcal{A}}\}$ with the highest validation score.
/* Obtain One-shot Mask */
Set $s_g\%$ of the lowest magnitude values in $\boldsymbol{m}_g^{\mathcal{A}}$ to 0 and others to 1, then obtain one-shot mask $\mathbf{M}_g^{\text{OS}}$.
/* Dynamically Update Edge Mask */
Set $\mathbf{M}_g^{(1)} \leftarrow \mathbf{M}_g^{\text{OS}}$.
for iteration $d \leftarrow 1$ **to** D **do**
  Compute interval index $\mu \leftarrow \lceil d/\Delta T \rceil$.
  Forward $f\left(\{\boldsymbol{m}_g \odot \mathbf{M}_g^{(\mu)} \odot \mathbf{A}, \mathbf{X}\}, \mathbf{\Theta}\right)$ to compute the $\mathcal{L}_{\text{anchor}}$.
  Update $\mathbf{\Theta}$ and $\boldsymbol{m}_g$ accordingly.
  /* Update Graph Structure */
  if $\mu = \lceil d/\Delta T \rceil$ **then**
    Set $\mathcal{E}_{(\mu)} \leftarrow$ edges in $\mathbf{M}_g^{(\mu)} \odot \mathbf{A}$, $\mathcal{E}_{(\mu)}^C \leftarrow$ edges in $\neg\mathbf{M}_g^{(\mu)} \odot \mathbf{A}$.
    for edge (i, j) in $\mathcal{E}$ **do**
      Compute semantic criteria $\phi^{(\text{sema})}(e_{ij}) \leftarrow \left|\nabla_{e_{ij}} \mathcal{S}(\mathcal{G}^{\mathcal{A}}, \mathcal{G}^{(\mu)})\right|$. ; ▷ Eq. 12
      /* Topology criterion only needs to be computed at the 1^{st} update. */
      Compute topological criteria $\phi^{(\text{topo})}(e_{ij}) \leftarrow \left(\sum_{k=1}^{K} + \sum_{k=N-K}^{N}\right) \frac{\boldsymbol{m}_{g,ij}^{\mathcal{A}} a_i^{(k)} b_j^{(k)}}{\lambda_k (\mathcal{G}^{\mathcal{A}}) a^{(k)T} b^{(k)}}$; ▷ Eq. 10
      Combine semantic and topological criteria $\phi(e_{ij}) \leftarrow \beta^s \cdot \phi^{(\text{sema})}(e_{ij}) + \beta^t \cdot \phi^{(\text{topo})}(e_{ij})$; ▷ Eq. 13
    end
    Compute the number of edges to be swapped $r \leftarrow \mathbf{\Upsilon}(\mu)$; ▷ Eq. 22
    /* Select Drop/Regrow Edges */
    Set $\mathbf{M}^{(\text{prune})} \leftarrow \text{ArgTopK}\left(-\phi(\mathcal{E}_{(\mu)}), r\right), \mathbf{M}^{(\text{regrow})} \leftarrow \text{ArgTopK}\left(\phi(\mathcal{E}_{(\mu)}^C),\ r\right)$; ▷ Eq. 5
    Update edge masks $\mathbf{M}_g^{(\mu+1)} \leftarrow \left(\mathbf{M}_g^{(\mu)} \setminus \mathbf{M}_g^{(\text{prune})}\right) \cup \mathbf{M}_g^{(\text{regrow})}$; ▷ Eq. 6
end

The memory complexity of GST is:

$$\mathcal{O}\left(\underbrace{L \times N \times D}_{\text{node embeddings}} + \overbrace{L \times |\mathbf{\Theta}| \times D^2}^{\text{GNN parameter}} + \underbrace{|\Psi| \times D}_{\text{graph masker}}\right) \tag{21}$$

E. Algorithm Framework

The algorithm framework is presented in Algo. 1.

F. Dataset Description

We conclude the dataset statistics in Tab. 7

Table 7. Graph datasets statistics.

Dataset	Nodes	Edges	Ave. Degree	Features	Classes	Metric
Cora	2,708	5,429	3.88	1,433	7	Accuracy
Citeseer	3,327	4,732	1.10	3,703	6	Accuracy
PubMed	19,717	44,338	8.00	500	3	Accuracy
Ogbn-ArXiv	169,343	1,166,243	13.77	128	40	Accuracy
Ogbn-Proteins	132,534	39,561,252	597.00	8	2	ROC-AUC
Ogbn-Products	2,449,029	61,859,140	50.52	100	47	Accuracy

G. Experimental Configurations

G.1. Train-val-test Splitting of Datasets.

To rigorously verify the effectiveness of our proposed GST, we unify the dataset splitting strategy across all GNN backbones and baselines. As for node classification tasks of small- and medium-size datasets, we utilize 420 (Citeseer) and 460 (PubMed) labeled data for training, 500 nodes for validation and 500 nodes for testing. For Squirrel and Chameleon datasets, we follow the original settings in (Cui & Wei, 2023; Rozemberczki et al., 2021), and set the train/valid/test ratio as 60%/20%/20%. The data splits for Ogbn-ArXiv, Ogbn-Proteins, and Ogbn-Products were provided by the benchmark (Hu et al., 2020). Specifically, for Ogbn-ArXiv, we train on papers published until 2017, validate on papers from 2018 and test on those published since 2019. For Ogbn-Proteins, protein nodes were segregated into training, validation, and test sets based on their species of origin. For Ogbn-Products, we use the sales ranking (popularity) to split nodes into training/validation/test sets. Concretely, we sort the products according to their sales ranking and use the top 8% for training, the next top 2% for validation, and the rest for testing.

G.2. Baseline Configurations

We detail how we report the results of baseline methods:

- Topology-based sparsification
  - **SCAN** (Spielman & Srivastava, 2008): SCAN uses structural similarity (called SCAN similarity) measures to detect clusters, hubs, and outliers. We utilize the implementation in (Chen et al., 2023)
  - **Local Similarity** (Hamann et al., 2016): Local Similarity ranks edges using the Jaccard score and computes $\log(\text{rank}(e_{ij}))/\log(\deg(e_{ij}))$ as the similarity score, and selects edges with the highest similarity scores. We utilize the implementation in (Chen et al., 2023).
  - **DSpar** (Liu et al., 2023): DSpar is an extension of effective resistance sparsifier, which aims to reduce the high computational budget of calculating effective resistance through an unbiased approximation. We adopt their official implementation (Liu et al., 2023).
- Semantic-based sparsification
  - **UGS** (Chen et al., 2021): We utilize the official implementation from the authors. Notably, UGS was originally designed for joint pruning of model parameters and edges. Specifically, it sets separate pruning parameters for parameters and edges, namely the weight pruning ratio p_θ and the graph pruning ratio p_g. In each iteration, a corresponding proportion of parameters/edges is pruned. For a fairer comparison, we set $p_\theta = 0\%$ and $p_g \in \{5\%, 10\%\}$ to get the results of all sparsity granularity.
  - **WD-GLT** (Hui et al., 2023): WD-GLT inherits the iterative magnitude pruning paradigm from UGS, so we also set $p_\theta = 0\%$ and $p_g \in \{5\%, 10\%\}$ across all datasets and backbones. The perturbation ratio α is tuned among $\{0, 1\}$. Since no official implementation is provided, we carefully reproduced the results according to the original paper.
  - **Meta-gradient sparsifier** (Wan & Schweitzer, 2021): The Meta-gradient sparsifier prunes edges based on their meta-gradient importance scores, assessed over multiple training epochs. Since no official implementation is provided, we carefully replicated the results following the guidelines in the original paper.

In addition to our selected baselines, we also enumerate other classic baselines relevant to graph sparsification and explain why they were not chosen for our study:

- **Effective Resistance Sparsifier** (Spielman & Srivastava, 2008): This method is one of the most renowned spectral sparsifiers. However, due to its high time complexity ($\prime(E\log(|\mathcal{V}|)^3)$), we opted for its approximate version, DSpar (Liu et al., 2023).
- **DropEdge** (Rong et al., 2019): Though DropEdge also involves dropping edges during training, the dropping process is random across different training epochs. Thus, it is not capable of outputting a compact yet performant subgraph, and we therefore do not consider it for comparison.
- **NeuralSparse** (Zheng et al., 2020): NeuralSparse is a well-recognized method for graph sparsification. Nevertheless, it cannot regulate the ultimate graph sparsity ratio. Consequently, we do not take it into consideration.
- **FastGCN** (Chen et al., 2018): FastGCN and other graph samplers (Chen et al., 2017; Ying et al., 2018a) samples neighbors for each in-batch node with a certain probability. However, the sampling process is only for training and does not essentially output a sparse graph.

G.3. Update Scheduler

$\boldsymbol{\Upsilon}(\mu)$ is the update scheduler which determines the number of edges to be swapped at each update. We simply adopt the Inverse Power (Zhu & Gupta, 2017):

$$\boldsymbol{\Upsilon}(\mu) = \tau(1 - \frac{\mu}{\lceil D/\Delta T \rceil})^{\kappa}, \tag{22}$$

where τ denotes the initial ratio and κ is the decay factor controlling how fast the ratio decreases with intervals.

G.4. Parameter Configurations

The main parameters of GST include: E (number of epochs to acquire the anchor graph), D (number of epochs to dynamically fine-tune the sparse graph), τ (the initial ratio of edges to swap), κ (the decay factor of $\boldsymbol{\Upsilon}(\mu)$), ΔT (the update interval), β^t (topology criterion coefficient), β^s (semantic criterion coefficient), and the learning rate.

We uniformly set τ to 0.3 and κ to 1. Ablation experiments regarding them are available in Appendix H.5. For small graphs, we employ 2-layer GCN/GIN/GAT, and for OGB graphs, we utilize 3-layer GraphSAGE and ClusterGCN (using GCN aggregation). Adhering to the principle of balancing attention to topological and semantic importance, we scale $\phi^{(\text{topo})}$ and $\phi^{(\text{sema})}$ to the same order of magnitude. Consequently, we uniformly set $\beta^t = 1$ and $\beta^s = 1e4$ across all experiments. The other hyperparameters are listed in Tab. 8.

Table 8. Hyper-parameter configurations.

Computing Infrastructures: NVIDIA Tesla V100 (32GB) Software Framework: Pytorch															
Param	Cora			Citeseer			PubMed			ArXiv		Proteins		Products	
	GCN	GIN	GAT	GCN	GIN	GAT	GCN	GIN	GAT	SAGE	ClusterGCN	SAGE	ClusterGCN	SAGE	ClusterGCN
E	100	100	100	100	200	200	200	200	200	200	100	200	200	200	200
D	400	400	400	500	500	500	600	600	600	300	300	300	300	300	300
lr	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.01	0.01	0.01	0.01	0.01	0.01
ΔT	20	20	20	20	20	20	20	20	20	3	3	3	3	3	3

G.5. Performance Metrics

Accuracy represents the ratio of correctly predicted outcomes to the total predictions made. The ROC-AUC (Receiver Operating Characteristic-Area Under the Curve) value quantifies the probability that a randomly selected positive example will have a higher rank than a randomly selected negative example. Hit@50 denotes the proportion of correctly predicted edges among the top 50 candidate edges.

G.6. Efficiency Metrics

To assess the efficiency of sparse graphs generated by various sparsifiers, we utilize two metrics: MACs (Multiply-Accumulate Operations) and GPU Inference Latency (ms). MACs theoretically represent the model's inference speed,

based on FLOPs (Floating Point Operations Per Second). Although `SpMM` is theoretically faster than `MatMul` based on MACs/FLOPs, this advantage is not always realized in practice due to `SpMM`'s random memory access pattern. To gauge the real-world applicability of our method, we additionally measure latency on GPUs in milliseconds (ms).

H. Additional Experimental Results

H.1. Experiments for $\mathcal{RQ}1$

This section details GST's performance on Cora, Ogbn-Proteins, and Ogbn-Products datasets. Fig. 7 shows the performance of GST with GCN/GIN/GAT. Notably, GST excels on Cora+GAT, experiencing only a negligible performance loss ($\approx 1.8\%$) at 60% graph sparsity.

Tabs. 9 and 10 present GST's performance on Ogbn-Proteins and Products. LSim, DSpar, and UGS were chosen as baselines due to the limitations of other baselines in inductive settings or scalability to large graphs. Generally, GST maintains superior performance across all sparsity levels. Specifically, it preserves GraphSAGE/ClusterGCN performance with negligible loss ($\leq 1\%$) at $10\% \sim 30\%$ sparsity. At 60% sparsity, GST outperforms LSim/DSpar by up to 6.95%.

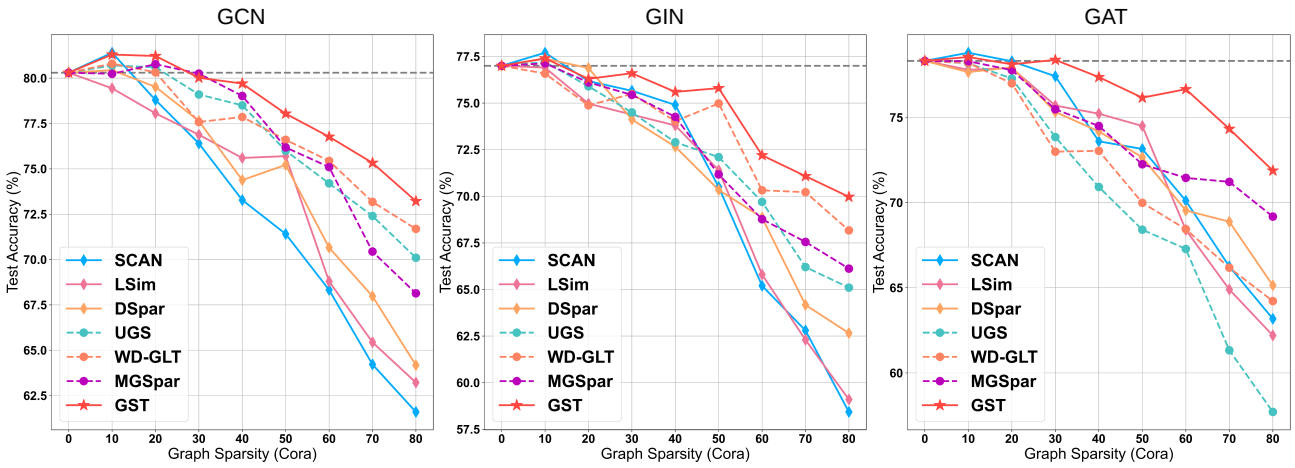

Figure 7. Performance comparison of graph sparsification methods on Cora under different sparsity levels. The grey dashed line represents the original baseline performance.

Table 9. Performance comparison of sparsifiers at different sparsity levels ($10\% \rightarrow 60\%$) on GraphSAGE/ClusterGCN with Ogbn-Proteins. We report the mean accuracy ± stdev of 3 runs. We shade the best results and underline the second-best results.

	GraphSAGE					
Sparsity	10%	20%	30%	40%	50%	60%
LSim	$76.92_{\pm 0.58}$	$75.03_{\pm 1.17}$	$73.20_{\pm 0.79}$	$73.69_{\pm 0.52}$	$72.14_{\pm 0.48}$	$70.09_{\pm 0.59}$
DSpar	$76.04_{\pm 0.25}$	$75.86_{\pm 0.27}$	$74.46_{\pm 0.39}$	$73.45_{\pm 0.26}$	$70.22_{\pm 0.54}$	$69.23_{\pm 0.40}$
UGS	$77.47_{\pm 0.27}$	$76.38_{\pm 0.63}$	$74.58_{\pm 0.49}$	$72.12_{\pm 0.38}$	$72.38_{\pm 0.33}$	$71.45_{\pm 0.75}$
GST	$77.59_{\pm 0.52}$	$77.56_{\pm 0.79}$	$76.45_{\pm 0.66}$	$76.02_{\pm 0.68}$	$75.45_{\pm 0.77}$	$73.55_{\pm 0.50}$
	ClusterGCN (GCN aggr)					
Sparsity	10%	20%	30%	40%	50%	60%
LSim	$75.44_{\pm 0.57}$	$74.18_{\pm 0.60}$	$74.27_{\pm 0.89}$	$72.15_{\pm 0.64}$	$69.42_{\pm 0.91}$	$66.46_{\pm 0.85}$
DSpar	$76.38_{\pm 0.79}$	$76.22_{\pm 1.13}$	$75.28_{\pm 0.85}$	$74.66_{\pm 1.05}$	$72.89_{\pm 0.74}$	$71.39_{\pm 0.92}$
GST	$76.29_{\pm 0.56}$	$76.11_{\pm 0.57}$	$76.16_{\pm 0.44}$	$75.83_{\pm 0.69}$	$73.03_{\pm 0.65}$	$73.80_{\pm 0.77}$

Table 10. Performance comparison of sparsifiers at different sparsity levels $\{10\%, 20\%, 30\%, 40\%, 50\%, 60\%\}$ on GraphSAGE/ClusterGCN with Ogbn-Products. Due to the immense scale of Ogbn-Products, we report results from a single run only. We shade the best results and underline the second-best results.

	GraphSAGE					
Sparsity	10%	20%	30%	40%	50%	60%
LSim	77.96	76.60	74.98	72.23	72.67	72.49
DSpar	<u>78.25</u>	<u>77.41</u>	<u>75.19</u>	<u>74.20</u>	<u>74.57</u>	<u>74.08</u>
GST	78.79	78.52	77.15	77.03	75.86	75.77
	ClusterGCN (GCN aggr)					
Sparsity	10%	20%	30%	40%	50%	60%
LSim	77.04	74.34	72.50	70.24	69.92	65.10
DSpar	<u>78.25</u>	<u>76.11</u>	<u>74.39</u>	<u>72.06</u>	<u>69.74</u>	<u>69.56</u>
GST	78.38	78.45	77.83	77.19	75.71	73.05

H.2. Experiments for $\mathcal{RQ}2$

In Figs. 4 and 8 to 10, we showcase the spectral preservation performance of GST, UGS, and LSim on Citeseer/PubMed with GCN/GIN. Specifically, we illustrate the distribution of relative error for the top-200 and bottom-200 eigenvalues at 20% and 50% sparsity levels produced by different methods. Our observations include: (1) GST significantly outperforms UGS/LSim in preserving the bottom-200 eigenvalues, which indicate local smoothness. For instance, on PubMed+GIN (in Fig. 10), at 50% sparsity, GST's average relative error is about 10% lower than UGS and 15% lower than LSim, demonstrating GST's unique advantage in maintaining the local smoothness of sparse graphs. (2) For the top-200 eigenvalues, GST performs best on larger graphs, notably PubMed, with PubMed+GCN/GIN (in Figs. 4 and 10) showing GST surpassing UGS/LSim for the top-200 eigenvalue preservation.

H.3. Experiments for $\mathcal{RQ}3$

In Figs. 5 and 11, we demonstrate GST's inference acceleration for GraphSAGE/ClusterGCN on OGB datasets. Due to UGS's inapplicability to the inductive ClusterGCN, values for UGS+ClusterGCN are omitted. It is observed that GST's inference acceleration on large-scale graphs is more pronounced than on small-scale graphs (as shown in Tab. 3). With negligible performance loss ($\leq 1\%$), GST achieves an inference speedup of $2.50 \sim 2.85\times$ on Ogbn-Products and $3.14 \sim 3.42\times$ on Ogbn-Proteins.

H.4. Experiments for $\mathcal{RQ}4$

As discussed in Sec. 4.5, we validate the versatility and plug-and-play nature of GST through two tasks: graph adversarial defense and graph lottery ticket identification.

Fig. 12 demonstrates how GST at sparsity levels of $\{10\%, 20\%, 30\%\}$ assists GCN/GAT in combating edge perturbations. Observations reveal that GST with proper sparsity significantly enhances GNN's resilience to edge perturbations. Across various datasets and backbones, the right level of graph sparsity notably improves GNN performance post-disturbance. For instance, on Cora+GCN, GST-10% recovers 7.23% of GCN's performance under a 30% perturbation ratio; on PubMed+GAT, GST-30% helps GAT regain 5.09% performance at a 40% perturbation ratio. This suggests that graphs of different sizes require varying degrees of sparsity for effective edge perturbation resistance.

Tab. 11 presents the performance when replacing UGS's iteratively pruned sparse graphs with GST-discovered graphs of the same sparsity. The results show that in most iterations, GST-discovered graphs significantly outperform UGS-located graph lottery tickets, demonstrating GST's broad applicability.

H.5. Experiments for $\mathcal{RQ}5$

In this section, we present detailed data for ablation study and parameter sensitivity analysis. Fig. 13 displays the performance of GST and its three variants on Citeseer+GCN/GIN and Ogbn-ArXiv+GraphSAGE/ClusterGCN. Tab. 13

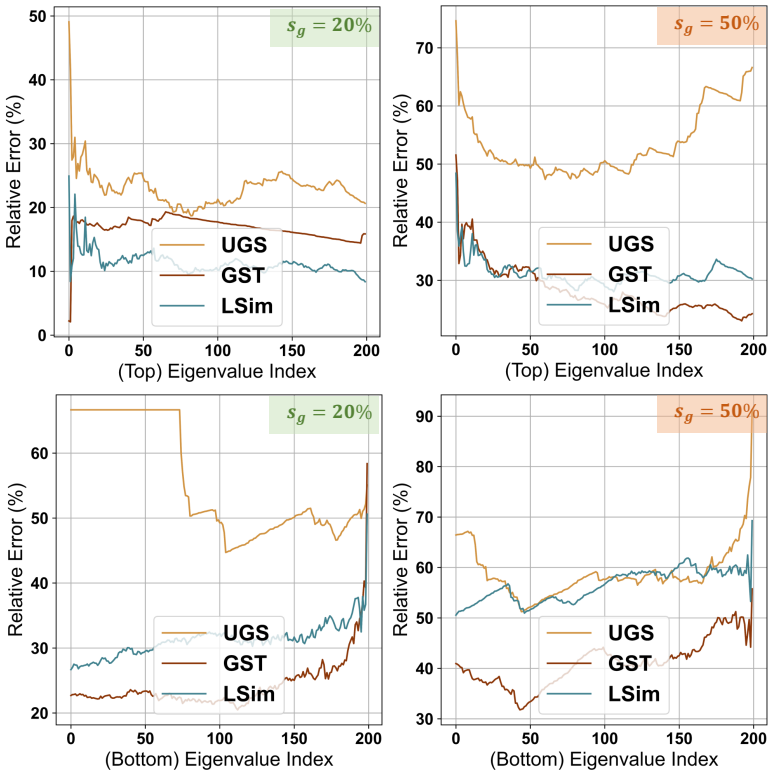

Figure 8. The relative error of the top-200 and bottom-200 eigenvalues on Citeseer+GCN, *i.e.*, $\frac{\lambda_i - \lambda_i'}{\lambda_i}$, sparsified by different methods at sparsity level 20% and 50%.

Table 11. Performance comparison of original UGS and UGS+GST on GCN backbone.

Weight Sparsity	20.0%	48.80%	67.23%	79.03%	86.58%	96.50%	98.95%
Graph Sparsity	5.0%	14.3%	22.6%	30.17%	36.98%	43.12%	48.67%
Dataset	**Cora**						
UGS	79.38	78.08	77.36	78.05	76.22	75.35	74.83
UGS+GST	80.25	79.21	78.83	77.66	77.24	76.38	75.62
Δ	0.87 ↑	1.13 ↑	1.47 ↑	0.39 ↓	1.02 ↑	1.03 ↑	0.79 ↑
Dataset	**Citeseer**						
UGS	70.30	69.77	68.14	69.02	68.53	67.59	66.84
UGS+GST	70.08	69.54	69.48	69.27	68.79	68.16	67.34
Δ	0.22 ↓	0.23 ↓	1.34 ↑	0.22 ↑	0.82 ↑	0.57 ↑	0.5 ↑
Dataset	**PubMed**						
UGS	78.51	77.21	75.60	75.17	74.85	75.21	74.96
UGS+GST	78.60	78.05	77.18	77.03	75.92	75.69	75.30
Δ	0.09 ↑	0.84 ↑	1.58 ↑	0.86 ↓	1.07 ↑	0.48 ↑	0.34 ↑

shows the performance of GST under various settings of α and κ, while Tab. 12 demonstrates how the performance of GST varies with ΔT.

Regarding ΔT, we can observe that smaller graphs (Citeseer) benefit from a lower update frequency, whereas larger graphs (Ogbn-ArXiv) perform better with more frequent updates. Specifically, $\Delta T = 20$ shows consistently good performance on Citeseer+GCN/GIN, and $\Delta T = 3$ excels on Ogbn-Arxiv. This observation is intuitive: larger graphs, with their more complex structures and a wider variety of potential sparse graph structures, necessitate more frequent structural updates. Therefore, we standardize the setting of $\Delta T = 20$ for small graphs and $\Delta T = 3$ for large graphs.

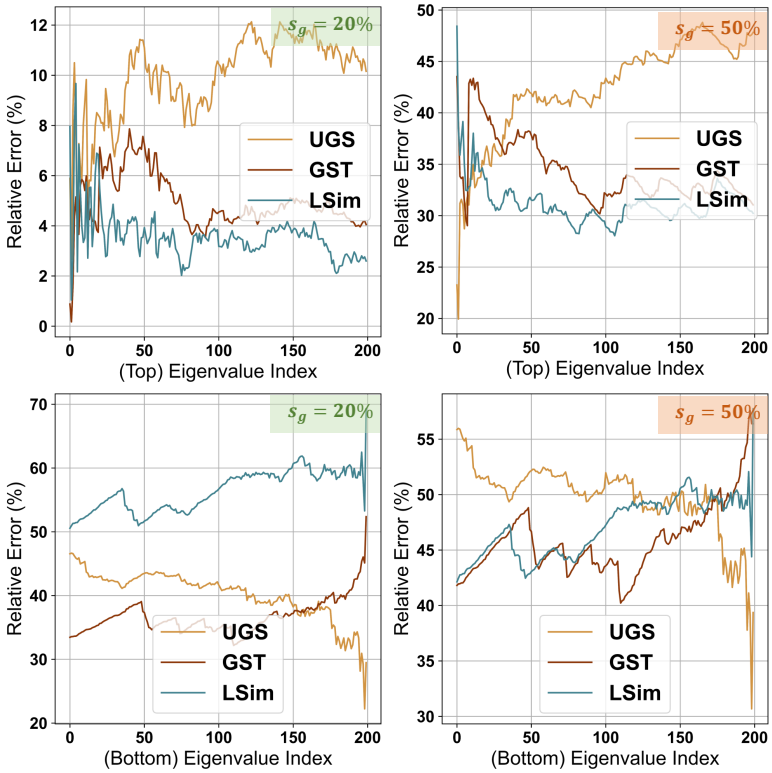

Figure 9. The relative error of the top-200 and bottom-200 eigenvalues on Citeseer+GIN, *i.e.*, $\frac{\lambda_i - \lambda_i'}{\lambda_i}$, sparsified by different methods at sparsity level 20% and 50%.

Regarding parameters τ and κ, it's clear that GST's performance is relatively insensitive to these choices, with fluctuations on GCN, GIN, and GAT not exceeding 1.69%, 1.88%, and 1.25% respectively. Overall, however, a larger α and a smaller κ, which both correspond to more frequent structural updates, tend to yield better performance.

Table 12. Ablation study on GST with its different ΔT. We report the extreme graph sparsity on Citeseer+GCN/GIN and Ogbn-ArXiv+GraphSAGE/ClusterGCN.

Dataset	Citeseer				Ogbn-ArXiv			
Backbone	GCN		GIN		GraphSAGE		ClusterGCN	
Sparsity	20%	50%	20%	50%	20%	50%	20%	50%
$\Delta T = 3$	69.16	68.01	69.15	66.02	71.80	68.02	69.21	67.33
$\Delta T = 5$	70.62	68.73	69.52	66.05	71.18	67.95	69.10	67.84
$\Delta T = 20$	70.89	68.46	69.71	66.16	68.04	67.77	69.40	66.25
$\Delta T = 50$	69.75	67.59	68.48	65.43	68.06	66.08	69.23	66.90

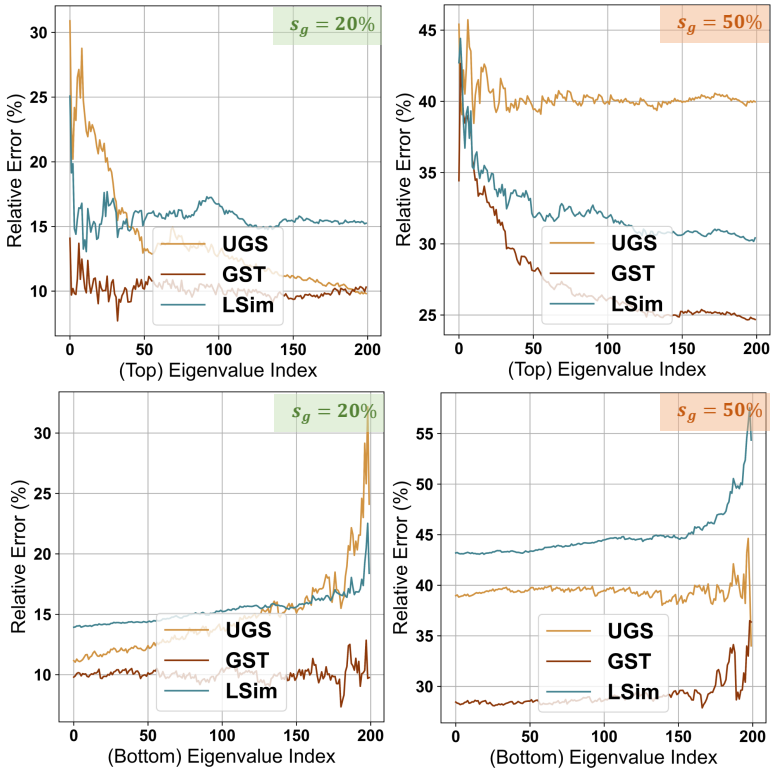

Figure 10. The relative error of the top-200 and bottom-200 eigenvalues on PubMed+GIN, *i.e.*, $\frac{\lambda_i - \lambda_i'}{\lambda_i}$, sparsified by different methods at sparsity level 20% and 50%.

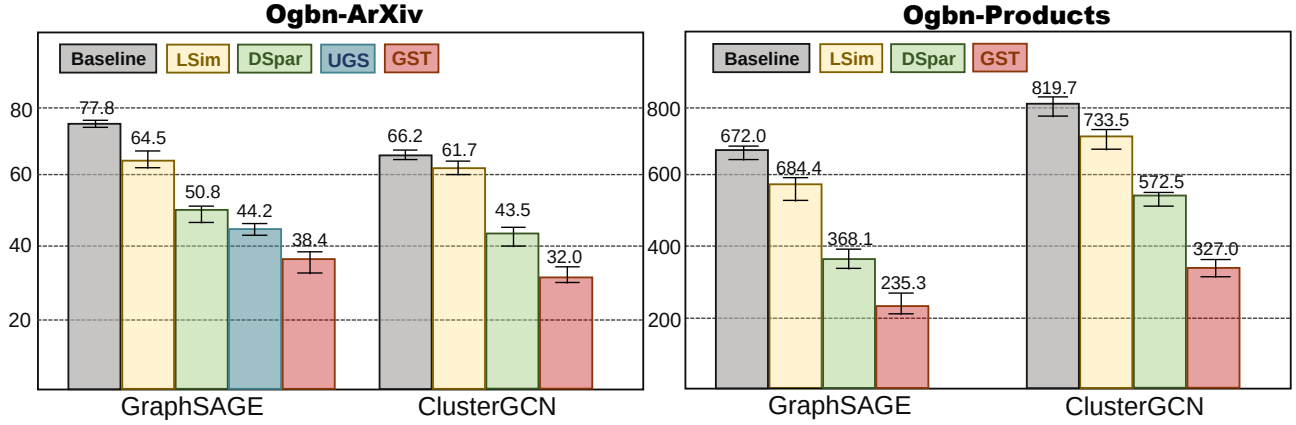

Figure 11. The inference latency on Ogbn-ArXiv/Products with different sparsifiers when their performance loss is negligible ($\leq 1\%$).

Table 13. Parameter sensitivity analysis on initial swapping ratio α and decay factor κ. We report the performance of GST on Citeseer dataset at 20% graph sparsity.

20%	GCN			GIN			GAT		
	$\alpha = 0.1$	$\alpha = 0.2$	$\alpha = 0.3$	$\alpha = 0.1$	$\alpha = 0.2$	$\alpha = 0.3$	$\alpha = 0.1$	$\alpha = 0.2$	$\alpha = 0.3$
$\kappa = 1$	69.32	70.86	70.89	67.64	68.98	69.35	68.03	68.42	68.55
$\kappa = 2$	69.14	70.84	69.85	67.25	70.86	70.89	67.66	68.12	68.33
$\kappa = 3$	68.26	68.30	69.20	67.48	69.11	69.20	67.50	67.17	67.95

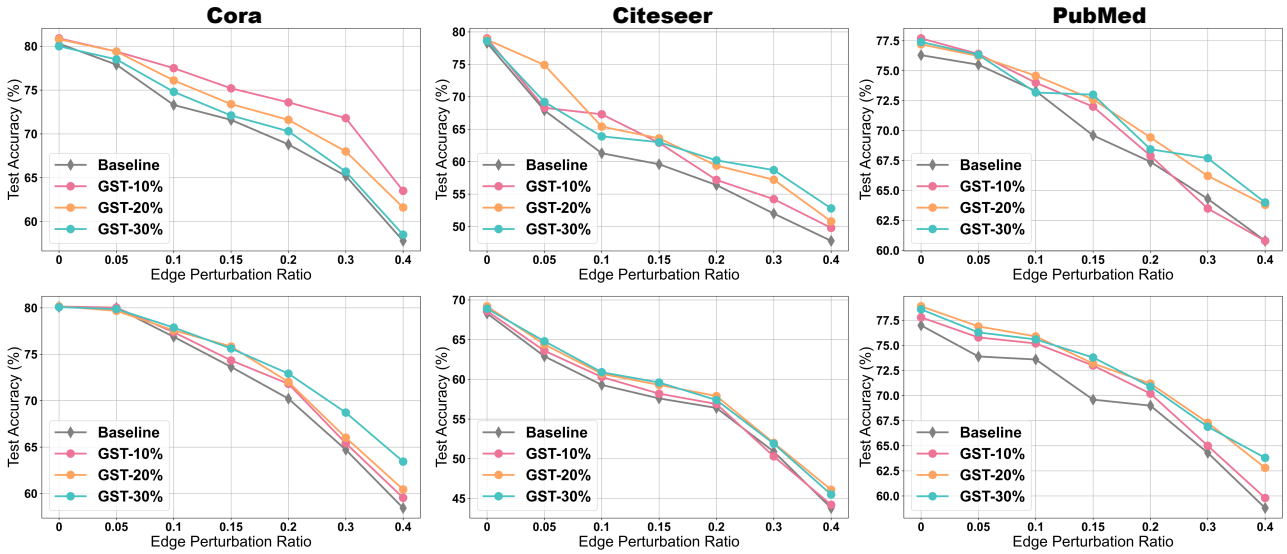

Figure 12. The robust performance of GST on edge perturbations by perturbing a varying fraction of edges $\{0\%, 5\%, 10\%, 15\%, 20\%, 30\%, 40\%\}$, tested on GCN (1st row) and GAT (2nd row).

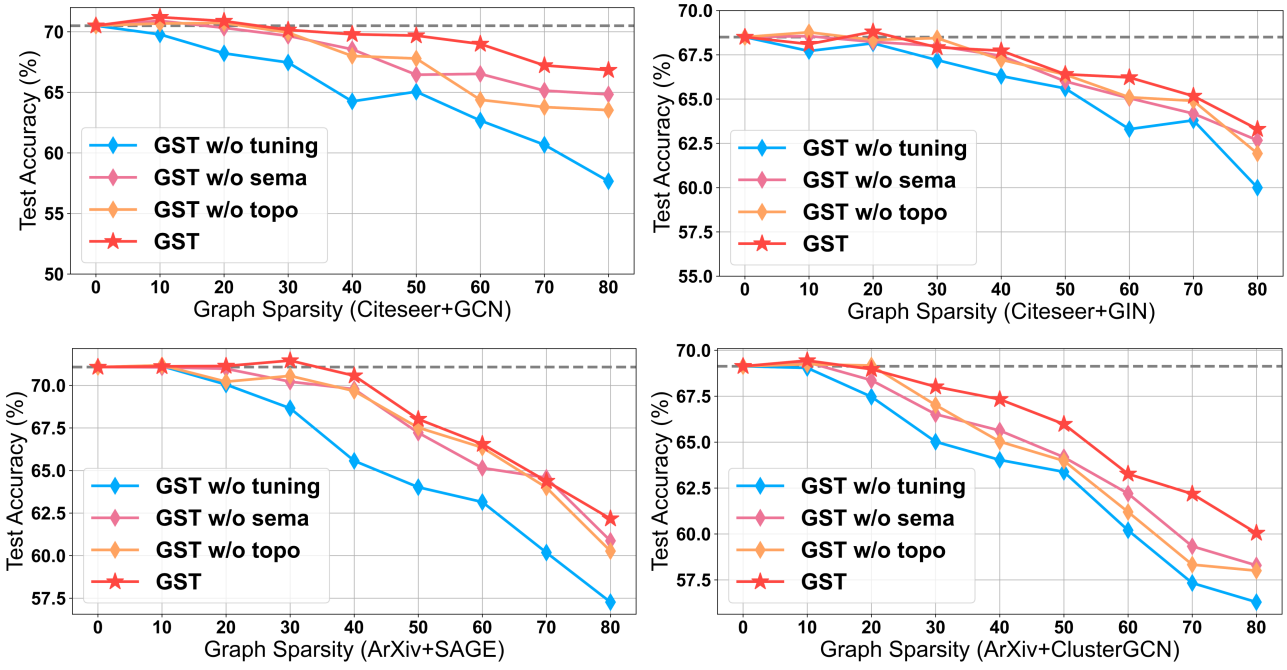

Figure 13. Ablation study of GST tested on Citeseer+GCN/GIN and Ogbn-ArXiv+GraphSAGE/ClusterGCN(GCN aggr).