SGS-GNN: A Supervised Graph Sparsifier for Graph Neural Networks

Anonymous

***Abstract*—We propose SGS−GNN, a novel supervised graph sparsifier that learns the sampling probability distribution of edges and samples sparse subgraphs of a user-specified size to reduce the computational costs required by GNNs for inference tasks on large graphs. SGS−GNN employs regularizers in the loss function to enhance homophily in sparse subgraphs, boosting the accuracy of GNNs on heterophilic graphs, where a significant number of the neighbors of a node have dissimilar labels. SGS−GNN also supports conditional updates of the probability distribution learning module based on a prior, which helps narrow the search space for sparse graphs. SGS−GNN requires fewer epochs to obtain high accuracies since it learns the search space of subgraphs more effectively than methods using fixed distributions such as random sampling. Extensive experiments using 33 homophilic and heterophilic graphs demonstrate the following: (i) SGS−GNN requires significantly less memory (up to $4\times$) compared to related baseline sparsification methods, (ii) with only 20% of edges retained in the sparse subgraphs, SGS−GNN improves the F1-scores by a geometric mean of 4% relative to the original graph; on heterophilic graphs, the prediction accuracy is better up to 30%. (iii) SGS−GNN outperforms state-of-the-art methods with improvement in F1-scores of $4-7\%$ in geometric mean with similar sparsities in the sampled subgraphs, and (iv) SGS−GNN is computationally efficient compared to related baselines and it requires about half the number of epochs to converge compared to supervised sparsifiers that sample from a fixed distribution.**

***Index Terms*—Graph Neural Networks, Graph Sparsification, Supervised Graph Sparsification.**

I. Introduction

We propose SGS−GNN, a supervised graph sparsifier that learns the sampling probability distribution of edges and samples sparse subgraphs to reduce the memory and computational costs of GNNs for inference tasks on large graphs. GNNs [1], [2] are tools for learning from graph-structured data, and are widely used in social networks, biological networks, computational physics, recommendation systems, etc. The two primary steps in a Graph Neural Network (GNN) are *aggregation* and *update*. For each node, the aggregation step involves accumulating embeddings from neighboring nodes using sparse matrix operations such as sparse-matrix dense-matrix multiplication (SpMM) and sampled dense-matrix dense-matrix multiplication (SDDMM) [3]. During the update step, a node updates its own embedding based on the aggregated information using dense matrix operations (MatMul) [3]. Approximately **70%** of the total cost is attributed to the SpMM operations [4]. Thus, by systematically removing nonessential edges, *graph sparsification* [5], [6] reduces memory and speeds up GNN training and inference while also improving accuracy.

A graph can be sparsified using either unsupervised and supervised methods. Unsupervised graph sparsification methods, such as spectral sparsification [7] and spanners [8], only focus on the structural characteristics of graphs, neglecting downstream tasks and the differences between homophilic (similar nodes connect) and heterophilic graphs (dissimilar nodes connect). This oversight can result in sparsified graphs that do not effectively support tasks such as node classification or link prediction [9]. In contrast, supervised graph sparsification methods [9] aim to reduce graph complexity while maintaining relevance to downstream tasks [10], [11]. Among supervised sparsification methods, SparseGAT [11] and SuperGAT [12] are considered *implicit sparsifiers* as they do not create a standalone sparse subgraph for independent use. Thus, they do not reduce the memory requirements of GNNs. In contrast, NeuralSparse [9] is an *explicit sparsifier* that constructs a subgraph based on a specified neighborhood size; however, this approach may lack precise control over sparsity and retain unnecessary edges. Furthermore, the vast search space for sparse subgraphs makes it challenging for supervised sparsifiers to find an optimal sampling distribution within a limited number of iterations.

To address these limitations, we propose SGS−GNN (§IV), a fast and lightweight supervised graph sparsifier that learns the edge probability distribution of an input graph to construct a sparse subgraph for downstream GNNs. SGS−GNN consists of three components: (i) edge probability encoding, (ii) sparse subgraph sampler, and (iii) downstream GNN. The edge probability encoding, called EdgePE, maps associated node features of edges into probabilities indicating the likelihood of specific edges to exist in the learned sparse subgraph. The subgraph sampler then samples a subgraph based on these learned probabilities. *Finally, the GNN can be any message-passing neural network.* SGS−GNN provides a unique advantage by allowing the downstream GNN to operate solely on a reduced sparse graph instead of the input graph, and supports batch processing for large graphs. In addition to the task-specific loss, one of the regularizers promotes homophily in sampled subgraphs, enhancing prediction quality in heterophilic graphs. The key contributions of our work are:

(I) We propose a novel graph sparsification technique, SGS−GNN, that works for both homophilic and heterophilic graphs (§IV). With SGS−GNN, users can control the amount of sparsity, efficiently learn sampling probabilities, and scale to large graphs with batch processing. SGS−GNN incorporates degree-proportionate edge weights as a *prior* sampling

distribution to guide the search for sparse subgraphs. The conditional update module in SGS−GNN encodes edge probabilities to further optimize performance and helps the trained model perform no worse than fixed distribution sparsifiers. We provide theoretical bounds for SGS−GNN, ensuring quality of the learned embeddings (§V).

(II) SGS−GNN requires significantly less memory than related supervised sparsifiers during training. We report several large problems where other methods run out of memory, where SGS−GNN succeeds in inference tasks (§VI-A).

(III) Extensive experiments on 33 benchmark graphs (21 heterophilic, 12 homophilic) show that SGS−GNN outperforms fixed-distribution sparsifiers in the majority of the benchmarks with up to 30% improvement in F1-scores using only 20% of the edges (§VI-B).

(IV) SGS−GNN is compared with 13 baseline methods, including scalable, non-scalable, and heterophilic GNNs. In similar settings, SGS−GNN outperforms sparsification-based GNN methods such as GraphSAINT [13], NeuralSparse [9], SparseGAT [11], MoG [3], DropEdge [14], with geometric mean improvement of $4-7\%$ in F1-scores (§VI-C).

(V) In terms of runtime, SGS−GNN is more efficient than NeuralSparse and MoG, and competitive with SparseGAT (§VI-D). Finally, SGS−GNN requires half the number of epochs to converge compared to fixed distribution sparsifiers (§VI-E).

II. Preliminaries

Let $\mathcal{G} \triangleq (\mathcal{V}, \mathcal{E}, \boldsymbol{X})$ be an undirected graph with its set of nodes and edges denoted by $\mathcal{V}$ and $\mathcal{E}$ respectively. The node feature matrix $\boldsymbol{X} \in \mathbb{R}^{|\mathcal{V}| \times F}$ contains node feature $\boldsymbol{x}_v \in \mathbb{R}^F$ as a row vector for every node $v \in \mathcal{V}$. The adjacency matrix $\boldsymbol{A}_{\mathcal{G}}$ of size $|\mathcal{V}| \times |\mathcal{V}|$ captures the neighborhood of each node in $\mathcal{G}$. In node classification, the goal is to predict a label $y_v \in C$ for each node $v \in \mathcal{V}$ among the $|C|$ possible class labels. The training uses labeled nodes $\mathcal{V}_L \subset \mathcal{V}$, while the unlabeled nodes $\mathcal{V}_U = \mathcal{V} \backslash \mathcal{V}_L$ are used for validation and testing. A single-layer Graph Convolutional Network (GCN) [15] is defined as:

$$\boldsymbol{H}^{(l+1)} = \sigma(\hat{\boldsymbol{A}}_{\mathcal{G}} \boldsymbol{H}^{(l)} \boldsymbol{W}^{(l)}), \tag{1}$$

where $\boldsymbol{H}^{(l)}$ represents the node embedding at layer l, with $\boldsymbol{H}^{(0)} = \boldsymbol{X}$, and $\boldsymbol{W}^{(l)}$ is the learnable weight matrix in layer l. $\hat{\boldsymbol{A}}_{\mathcal{G}}$ denotes the normalized adjacency matrix and σ is an activation function such as ReLU. The predicted probabilities can be expressed as $\tilde{\boldsymbol{Y}} = \texttt{Softmax}(\texttt{GNN}_\theta(\mathcal{G}))$ where $\texttt{GNN}_\theta$ is a GNN with L layers and learnable parameters θ. The dimension of $\tilde{\boldsymbol{Y}}$ is $|\mathcal{V}| \times |C|$. The training objective is to find parameters θ that minimize the cross-entropy loss,

$$\mathcal{L}_{\text{CE}} = -\frac{1}{|\mathcal{V}_L|} \sum_{v \in \mathcal{V}_L} \sum_{c=1}^{|C|} Y_{vc} \log \tilde{Y}_{vc}, \tag{2}$$

where Y_{vc} indicates the true probability of node v belonging to class $c \in C$.

The *homophily* of a graph characterizes the likelihood that nodes with the same labels are neighbors. Some well-known definitions include *Node homophily* $\mathcal{H}_n$ [16], *Edge homophily* $\mathcal{H}_e$ [17], and *Adjusted Homophily* $\mathcal{H}_{\text{adj}}$ [18]:

$$\mathcal{H}_n = \frac{1}{|\mathcal{V}|} \sum_{u \in \mathcal{V}} \frac{|\{v \in \mathcal{N}(u) : y_v = y_u\}|}{|\mathcal{N}(u)|} \tag{3}$$

$$\mathcal{H}_e = \frac{(u,v) \in \mathcal{E} : y_u = y_v}{|\mathcal{E}|} \tag{4}$$

$$\mathcal{H}_{\text{adj}} = \frac{\mathcal{H}_e - \sum_{k=1}^{c} D_k^2/(2|\mathcal{E}|^2)}{1 - \sum_{k=1}^{c} D_k^2/2|\mathcal{E}|^2}. \tag{5}$$

Here $D_k = \sum_{v:y_v=k} d_v$ denotes the sum of degrees of the nodes belonging to class k. The values of the node homophily and the edge homophily range from 0 to 1, and the adjusted homophily ranges from $-\frac{1}{3}$ to $+1$. In this work, we classify graphs with $\mathcal{H}_{\text{adj}} \leq 0.50$ as heterophilic.

A. Problem Statement and Theoretical Motivation

Given $q > 0$, this paper aims to construct a sparse subgraph $\tilde{\mathcal{G}} \triangleq (\mathcal{V}, \tilde{\mathcal{E}}, \boldsymbol{X})$ with $q\%$ of original edges in $\mathcal{E}$. Let $\mathcal{G}_q$ be the space of all distinct sparse subgraphs of $\mathcal{G}$ where every subgraph contains exactly $k = \lfloor \frac{q|\mathcal{E}|}{100} \rfloor$ edges, $\mathcal{G}_q = \{\tilde{\mathcal{E}} \subset \mathcal{E} : |\tilde{\mathcal{E}}| = k\}$. The objective of our supervised sparse graph construction is to find the parameters θ along with a sparse subgraph $\tilde{\mathcal{G}} \in \mathcal{G}_q$ that minimizes $\mathcal{L}_{\text{CE}}$.

We can define a probability space $(\Omega, \mathcal{F}, p)$ by considering $\mathcal{E}$ as the sample space, $\mathcal{F} = \mathcal{G}_q \subseteq 2^\Omega$ as the event space and a suitable probability measure $p : \mathcal{F} \to [0,1]$. The probability measure is determined by which subgraphs result in a node representation that minimizes the loss in Eq. 2, which, in turn, depends on the downstream task. As a result, it is unknown which probability distribution is suitable as a choice for p. Specifically, we can perform the following decomposition to predict the probability that a node v belongs to a class $c \in C$,

$$P(\tilde{Y}_{vc}|\mathcal{G}) = \sum_{\tilde{\mathcal{G}} \in \mathcal{G}_q} P(\tilde{Y}_{vc}|\tilde{\mathcal{G}}) P(\tilde{\mathcal{G}}|\mathcal{G}). \tag{6}$$

There are two issues with the above decomposition. First, it requires enumerating all possible candidate subgraphs $\tilde{\mathcal{G}} \in \mathcal{G}_q$. This is computationally challenging because there are $\binom{|\mathcal{E}|}{k}$ subgraphs, and it is not possible to estimate the probabilities $P(\tilde{Y}_{vc}|\tilde{\mathcal{G}})$, $P(\tilde{\mathcal{G}}|\mathcal{G})$ due to their dependence on the downstream task under consideration.

We address these issues by encoding $P(\tilde{\mathcal{G}}|\mathcal{G})$ as a learnable neural network module $\texttt{EdgePE}_\phi$ that explicitly learns parameters ϕ to predict $\tilde{p} = \texttt{EdgePE}_\phi(\mathcal{G})$ which estimates the probability measure p for every edge based on the downstream task. The neural network searches the space $\mathcal{G}_q$ by adjusting its learned probability estimate $\tilde{p}$ based on the gradient of the loss. Finally, we model $P(\tilde{Y}_{vc}|\tilde{\mathcal{G}})$ as a GNN that takes the sparsified sample $\tilde{\mathcal{G}}$ sampled from the learned distribution $\tilde{p}$. Hence, Eq. 6 can be approximated by,

$$P(\tilde{Y}_{vc}|\mathcal{G}) \approx \mathbb{E}_{\tilde{\mathcal{G}} \sim \texttt{EdgePE}_\phi(\mathcal{G})}[\texttt{GNN}_\theta(\tilde{\mathcal{G}}) \cdot \texttt{EdgePE}_\phi(\mathcal{G})]. \tag{7}$$

Equation 7 follows since instead of directly searching over the space $\mathcal{G}_q$, we rely on the distribution $\tilde{p}$ approximated via a neural network. Once this distribution is learned, the law of

large numbers indicates that a sufficient number of samples from $\tilde{\mathcal{G}} \sim \tilde{p}$ can estimate $P(\tilde{Y})$. Thus, the summation can be replaced with the expected value from a learned GNN model using sparse subgraph samples.

III. Related Works

Unsupervised Graph Sparsification. Effective resistance (ER) [19] based sampling generates sparse subgraphs whose Laplacian eigenvalues are bounded by those of the original graph. FastGAT [20] uses ER to improve GNN efficiency, but the high computational cost of ER makes it impractical for large graphs. The random sparsifier is the fastest approach to get sparse subgraphs and is widely used in GNNs such as DropEdge [14] and GraphSAGE [21]. GraphSAINT [13] uses the normalized *degree* as edge weight to assign low sampling probability to edges in denser clusters. Later, it is used for sampling subgraphs and often produces better results than random sparsifiers. Spanner (e.g., t-spanner) [8] is a topology-based sparsifier that bounds distances between nodes in a sparse subgraph by at most a factor of t higher than the distance in the original graph. Spanning Tree and Forest are useful sparsifiers, as they preserve node connectivity, which helps message propagation in GNNs. Although both lack control over sparsity, the notion of connectivity is essential. Recently, Lottery Ticket Hypothesis (LTH) based sparsification methods [22] have gained popularity, but they are often impractical due to high computational complexity.
Another class of unsupervised sparsifiers first computes similarities between two nodes as edge weights and then samples according to those weights. It could be structural similarity, such as *Jaccard distance* on a portion of shared neighbors (SCAN [23]) or feature similarity (AGS-GNN [24]).
Supervised Graph Sparsification. While supervised graph sparsification can introduce computational overhead during the training phase, it is frequently offset by enhanced prediction accuracy, particularly in graphs that are noisy or heterophilic. Methods like SparseGAT [11] and SuperGAT [12] use the entire graph information during GNN training and learn sparse subgraphs through regularizers. SGCN [25] is another sparsification method that optimizes the runtime by alternating the sparsifier's and GNN's learning. NeuralSparse [9], PDTNet [10], and AD-GCL [26] explicitly learn to sample sparse subgraphs. Our `SGS-GNN` falls into a similar category with some distinctions. NeuralSparse samples k neighbors from each node to generate the sparse graphs, whereas `SGS-GNN` globally learns the sampling distribution of all edges and then samples from that distribution, which is significantly faster. AD-GCL samples subgraphs considering different objective functions. Unlike SGS-GNN, these methods use a mask matrix of the same size as the original adjacency matrix for gradient flow, which corresponds to implicit sparsification since they do not reduce the memory needed for **SPMM** operations. `SGS-GNN` uses a prior probability distribution to narrow the sparse subgraph search space; this notion of prior is useful [27] and recent work like Mixture of Graphs (MoG) [3] uses *Jaccard similarity*, *gradient magnitude*, and *effective resistance* as a prior for sparse subgraph selection. Additionally, LAGCN [28] employs an edge classifier to modify graphs based on training nodes, while our `EdgePE` coupled with regularizer uses training edges to foster homophily in the sampled subgraph.

IV. Proposed Method: SGS-GNN

Fig. 1 illustrates the inductive bias for edges with low sampling probability in the learned distribution. (I) The method `SGS-GNN` employs a prior (p_{prior}) that favors the removal of non-bridge edges (bridge edges connect different clusters). (II) It also discards edges that are irrelevant to the downstream task, using Loss $\mathcal{L}_{CE}$ based on the training nodes (represented by nodes with red circular borders in the figure). (III) A regularizer ($\mathcal{L}_{assor}$) decreases the sampling probability for heterophilic edges. (IV) Another regularizer ($\mathcal{L}_{cons}$) ensures consistency between embedding distances and edge weights, which reduces the selection probability for vertices with distant embeddings.

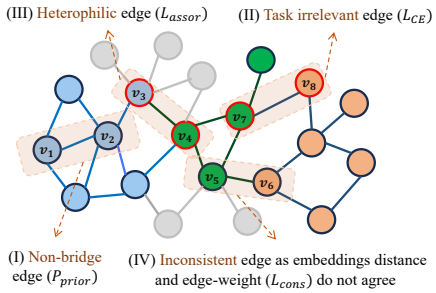

Fig. 1. Illustration of inductive bias from `SGS-GNN`, showing how prior and loss functions lower selection probability for certain edges. Nodes with red borders are training nodes.

In the following sections, we will discuss how this selection strategy is implemented in `SGS-GNN`. Fig. 2 illustrates the schematic of `SGS-GNN` and its primary components.

A. Module I: Edge Probability Encoding

Given input $\mathcal{G}$, the edge probability encoding module (`EdgePE`) maps the node features to edge weights in the range $[0, 1]$, followed by normalization to turn the learned weights into probabilities. The learned edge weights represent the model's un-normalized confidence in the existence of each edge. `EdgePE` learns the edge weights of (u, v) as a function of node embeddings $\boldsymbol{h}_u, \boldsymbol{h}_v$:

$$w(e_{uv}) = \sigma(\texttt{MLP}_\phi((\boldsymbol{h}_u - \boldsymbol{h}_v) \oplus (\boldsymbol{h}_u \odot \boldsymbol{h}_v))). \tag{8}$$

Here, σ refers to `Sigmoid` activation function, $\oplus$ indicates concatenation, $\odot$ represents element-wise multiplication, and $\texttt{MLP}_\phi$ is a Multi Layer Peceptron (`MLP`) learnable weights of ϕ. Let us assume $\boldsymbol{h}_u$ indicates the node embedding in matrix $\boldsymbol{H}$ corresponding to node u. Thus, the node embedding matrix $\boldsymbol{H}$ can be computed from an `MLP` in the following manner: $\boldsymbol{H} = \texttt{ReLU}(\texttt{MLP}_{\boldsymbol{W}}(\boldsymbol{X}))$, where $\texttt{MLP}_{\boldsymbol{W}}$ is an `MLP` with weights $\boldsymbol{W}$. `MLP` is computationally efficient; however, it

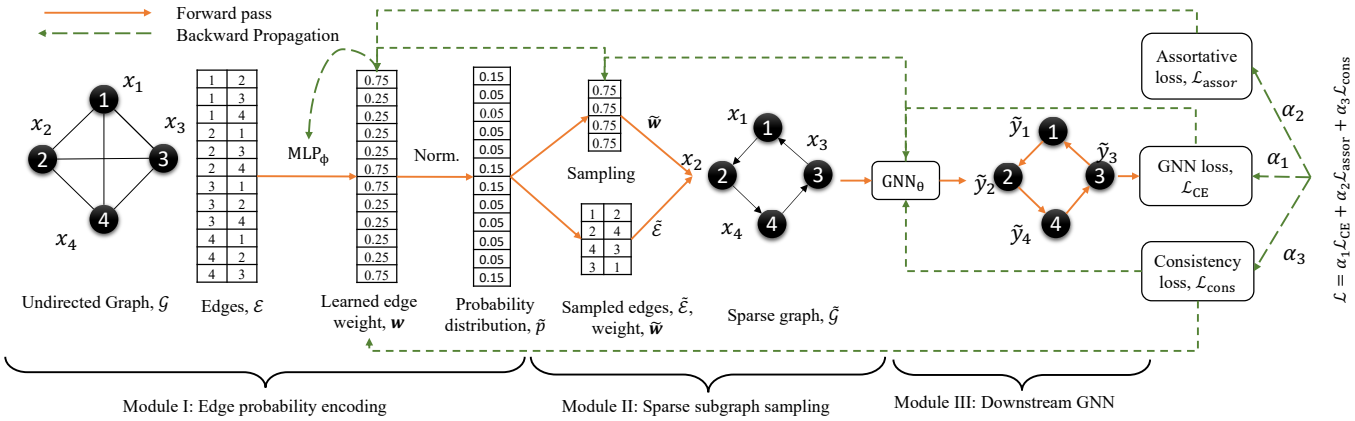

Fig. 2. Illustration of the three modules in SGS-GNN. The edge probability encoding module (EdgePE) computes a probability distribution, the sampler module samples the subgraph, and the downstream GNN makes predictions using that sparse subgraph.

does not exploit the graph structural information. As a result, $\texttt{MLP}_{\boldsymbol{W}}$ is not necessarily the most effective choice, as we show later in the ablation studies (§VI-G). A common approach to incorporating graph structural information is to utilize graph convolutions, such as vanilla GCN [15] or SAGE convolution [21]. For instance, one can compute a graph-structure aware node embedding matrix using a single-layer GCN as follows: $\boldsymbol{H} = \texttt{GCN}_{\boldsymbol{W}}(\mathcal{G}) = \sigma(\hat{\boldsymbol{A}}_{\mathcal{G}}\boldsymbol{X}\boldsymbol{W})$, where $\texttt{GCN}_{\boldsymbol{W}}$ is a GCN with learnable weights $\boldsymbol{W}$. However, considering the entire graph $\boldsymbol{A}_{\mathcal{G}}$ is memory-intensive for large graphs.

Thus SGS-GNN takes a subset of edges $\mathcal{E}_{\text{sp}} \subseteq \mathcal{E}$ of size $\lfloor \frac{q|\mathcal{E}|}{100}) \rfloor$ following a fixed prior probability distribution p_{prior} and uses the induced subgraph $\mathcal{G}[\mathcal{E}_{\text{sp}}]$ for computing node embedding $\boldsymbol{H}$. In order to maintain good connectivity in $\mathcal{G}[\mathcal{E}_{sp}]$, the prior distribution is defined for edges (u, v) as $p_{\text{prior}}(u, v) \propto (1/d_u + 1/d_v)$, where d_u, d_v are the degrees of nodes u, v. This notion of degree-based prior comes from Corollary 3.3 in Lovász [29], which gives a bound to the effective resistance for every $(u, v) \in \mathcal{E}$ as follows: $1/2 \cdot (1/d_u + 1/d_v) \leq R_e \leq 1/\alpha \cdot (1/d_u + 1/d_v)$, where $\alpha \leq 2$ is the smallest non-zero eigenvalue of $\boldsymbol{L} = \boldsymbol{I} - \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$.

Normalization. Normalization turns the learned edge weights into a valid probability distribution. One simple choice is *sum-normalization* computed as follows: $\tilde{p}(e_{uv}) = w(e_{uv})/\sum_{(u,v)\in\mathcal{E}} w(e_{uv})$. Another choice is *softmax-normalization* with temperature annealing,

$$\tilde{p}(e_{uv}) = \frac{\exp(w(e_{uv})/T)}{\sum_{(u,v)\in\mathcal{E}} \exp(w(e_{uv})/T)}. \tag{9}$$

Here, $T > 0$ is the temperature parameter. When T is large, the learned distribution $\tilde{p}$ approaches a uniform distribution over edges, whereas it approaches a categorical distribution when T is small [30]. As the learned distribution approaches a uniform distribution, the model tends to explore more diverse subgraphs from the subgraph space $\mathcal{G}_q$. On the other hand, as the learned distribution approaches a categorical distribution, the model tends to explore less in $\mathcal{G}_q$. Hence, we vary T as a function of training iterations such that in early iterations, the algorithm explores more while narrowing down to its preferred search space later on. We execute such an annealing mechanism with the following equation:

$$T = \max(T_{\min}, T_0 - \text{epoch} \cdot r), \tag{10}$$

where $r = (T_0 - T_{\min})/\text{max\_epochs}$ is the annealing rate, $T_{\min}$ is the minimum allowable temperature, and T_0 is the initial temperature. The temperature linearly decreases from the initial value T_0 to its final value $T_{\min}$ with the epochs. We keep track of the T-value that gives the best validation accuracy and use it later during inference.

Alg. 1 shows the pseudocode for EdgePE.

Algorithm 1 EdgePE Module

Input: $\mathcal{G}(\mathcal{V}, \mathcal{E}, \boldsymbol{X})$, sample % q, #layers L, temperature T
$\forall_{(u,v)\in\mathcal{E}}\ p_{\text{prior}}(u, v) \leftarrow \frac{1/d_u + 1/d_v}{\sum_{i,j\in\mathcal{E}}(1/d_i + 1/d_j)}$
$\mathcal{E}_{\text{sp}} \leftarrow \text{Multinomial}(\mathcal{E}, p_{\text{prior}}, \lfloor \frac{q|\mathcal{E}|}{100} \rfloor)$
$\boldsymbol{H} \leftarrow \texttt{GCN}_{\boldsymbol{W}}(\mathcal{E}_{\text{sp}}, \boldsymbol{X}, L)$
$\forall_{(u,v)\in\mathcal{E}}\ \boldsymbol{w}(u, v) = \sigma(\texttt{MLP}_\phi((\boldsymbol{h}_u^{(i)} - \boldsymbol{h}_v^{(i)}) \oplus (\boldsymbol{h}_u^{(i)} \odot \boldsymbol{h}_v^{(i)}))$
$\tilde{p} \leftarrow \text{Softmax}(\boldsymbol{w}/T)$
Return $\tilde{p}, \boldsymbol{w}$

B. Module II: Sparse Subgraph Sampling

Given the learned distribution $\tilde{p}$ over the edges of the input graph, sparse subgraph sampling aims to construct a sparse graph with the user-given sparsity constraint q. We do not know which discrete distribution has $\tilde{p}$ as parameters. A natural choice is to construct $\tilde{\mathcal{G}} = (\mathcal{V}, \tilde{\mathcal{E}}, \boldsymbol{X})$ by assuming that $\tilde{p}$ is a parameter of a *Multinomial* distribution. Hence we can sample $k = \lfloor \frac{q|\mathcal{E}|}{100} \rfloor$ edges as $\tilde{\mathcal{E}} \sim \text{Multinomial}(\tilde{p}, k)$.

We can also construct $\tilde{\mathcal{G}}$ by assuming that $\tilde{p}$ is a parameter of some categorical distribution and use the *Gumbel Softmax trick* [30]. The idea is to induce *Gumbel noise* $g_{uv} \sim Gumbel(0, 1)$ to the edges and select Top-K edges with

the highest probabilities. To sample edges from the categorical distribution, the edge probability is defined as:

$$\tilde{p}(e_{uv}) = \frac{\exp((\log w(e_{uv}) + g_{uv})/T)}{\sum_{(u,v)\in\mathcal{E}} \exp((\log w(e_{uv}) + g_{uv})/T)}. \quad (11)$$

Adding noise ensures that we are taking different samples at each time, and with low temperatures ($T = 0.1, 0.5$), the samples are identical to those from a categorical distribution [30].

C. Module III: Downstream GNN and Loss Functions

At this stage, we input the sampled subgraph to a downstream GNN that supports edge weights as computed in Eq. 8; since the edge weights of the sampled edges are one of the ways we optimize EdgePE via backpropagation. Thus, to keep the edge weights in our computation graph, we conduct our GNN's forward pass as follows:

$$\tilde{\boldsymbol{Y}} = \texttt{Softmax}(\texttt{GNN}_\theta(\mathcal{V}, \tilde{\mathcal{E}}, \boldsymbol{X}, \tilde{\boldsymbol{w}})), \quad (12)$$

where $\tilde{\mathcal{E}}$ refers to the edges of the sampled sparse subgraph $\tilde{\mathcal{G}}$ and $\tilde{\boldsymbol{w}} = \boldsymbol{w}[\tilde{\mathcal{E}}]$ contains the sampled edge weights.

Loss functions. We introduce two regularizers to engrain various inductive biases to SGS-GNN and combine these functions with the Cross-Entropy loss $\mathcal{L}_{\text{CE}}$ (Case II in Fig. 1) as follows:

$$\mathcal{L} = \alpha_1 \mathcal{L}_{\text{CE}} + \alpha_2 \mathcal{L}_{\text{assor}} + \alpha_3 \mathcal{L}_{\text{cons}}, \quad (13)$$

where $0 \leq \alpha_1, \alpha_2, \alpha_3 \leq 1$ are regularizer coefficients.

The **Assortativity loss** $\mathcal{L}_{\text{assor}}$ uses the labels of the training nodes to force nodes with similar labels to have higher edge weights while forcing dissimilarly labeled nodes to have a small nonzero weight. This regularizer encourages edge homophily in the sampled sparse graph (Case III in Fig. 1).

$$\mathcal{L}_{\text{assor}} \triangleq - \sum_{(u,v)\in\mathcal{E}: u\in\mathcal{V}_L \wedge v\in\mathcal{V}_L} \mathbb{I}(y_u = y_v) \cdot \log w(e_{uv}), \quad (14)$$

where $\mathbb{I}(.)$ is an indicator function that returns 0 or 1.

The **Consistency loss** defined below encourages learned edge probabilities to reflect the similarity between node embeddings or features:

$$\mathcal{L}_{\text{cons}} \triangleq \sum_{(u,v)\in\tilde{\mathcal{E}}} \|w(e_{uv}) - \text{cosine}(\boldsymbol{h}_u^l, \boldsymbol{h}_v^l)\|, \quad (15)$$

where $\text{cosine}(\boldsymbol{h}_u^l, \boldsymbol{h}_v^l) = \boldsymbol{h}_u^l \cdot \boldsymbol{h}_v^l / \|\boldsymbol{h}_u^l\| \|\boldsymbol{h}_v^l\|$ is the cosine similarity of the learned GNN embeddings $\boldsymbol{h}_u^l, \boldsymbol{h}_v^l$ of nodes u, v from layer l, and $w(e_{uv})$ is the learned probability for edge (u, v) in the sparse graph $\tilde{\mathcal{G}}$. This mechanism aligns the edge probabilities with the global graph structure and ensures that the sparsifier learns to preserve edges consistent with the broader graph relationships (Case IV in Fig. 1).

D. SGS-GNN Training and Additional Details

Alg. 2 outlines the pseudocode for training SGS-GNN. It starts with two precomputation steps: i) computing the degree-proportionate edge weight as a *prior* to enhance the learned distribution $\tilde{p}$ (line 1), and ii) partitioning the input graph using METIS [31] for batch processing (line 2). Towards computing the loss for every partition at each iteration, SGS-GNN executes edge probability encoding, learned distribution augmentation with a prior, sparse subgraph sampling, and node embedding via GNN. Finally, the loss is backpropagated, the update pathways of which have been illustrated in Figure 2.

Algorithm 2 SGS-GNN Training

1: **Input:** $\mathcal{G}(\mathcal{V}, \mathcal{E}, \boldsymbol{X})$, sample % q, #layers L, METIS Parts n
2: $p_{\text{prior}}(u, v) \leftarrow \frac{1/d_u + 1/d_v}{\sum_{i,j\in\mathcal{E}}(1/d_i + 1/d_j)}$
3: $\mathcal{G}_{\text{parts}} \leftarrow \{\mathcal{G}_1, \mathcal{G}_2, \cdots, \mathcal{G}_n\} = \text{METIS}(\mathcal{G}(\mathcal{V}, \mathcal{E}, p_{\text{prior}}), n)$
4: **for** epoch $= 1, 2, \ldots,$ max_epochs **do**
5: **for** $\mathcal{G}_i(\mathcal{V}_i, \mathcal{E}_i, \boldsymbol{X}_i, p^i_{\text{prior}}) \in \mathcal{G}_{\text{parts}}$ **do**
6: $\tilde{p}, \boldsymbol{w} \leftarrow \texttt{EdgePE}_\phi(\mathcal{E}_i, \boldsymbol{X}_i, L)$ {**Algorithm 1**}
7: $\tilde{p}_a \leftarrow \lambda\tilde{p} + (1-\lambda)p^i_{\text{prior}}$/*Augmenting $\tilde{p}$ with prior*/
8: $\tilde{\mathcal{E}}, \tilde{\boldsymbol{w}} \leftarrow \text{Sample}(\tilde{p}_a, \boldsymbol{w}, \lfloor \frac{q|\mathcal{E}|}{100} \rfloor)$ {**Module II**}
9: $\tilde{\boldsymbol{Y}}, \tilde{\boldsymbol{H}} \leftarrow \texttt{GNN}_\theta(\tilde{\mathcal{E}}, \boldsymbol{X}_i, \tilde{\boldsymbol{w}})$ {**Module III**}
10: Compute $\mathcal{L}_{CE}, \mathcal{L}_{\text{assor}}$, and $\mathcal{L}_{\text{cons}}$ using $\tilde{\boldsymbol{Y}}, \tilde{\boldsymbol{H}}$
11: $\mathcal{L} \leftarrow \alpha_1 \cdot \mathcal{L}_{\text{CE}} + \alpha_2 \cdot \mathcal{L}_{\text{assor}} + \alpha_3 \cdot \mathcal{L}_{\text{cons}}$
12: Backpropagate through $\mathcal{L}$ to update $\texttt{EdgePE}_\theta, \texttt{GNN}_\theta$
13: **end for**
14: **end for**

Batch processing. We can use EdgePE from Alg. 1 to compute edge weights in large-scale graphs, but efficient batch processing on edges is necessary for stochastic training of GNNs so as to reduce the risk of getting stuck in local minima. It is crucial to select a batch of edges that have high locality, preferably from within a cluster, and we utilize METIS to achieve this. We could have made partitions small enough to fit GPUs and then applied GNN without any sparsification, similar to ClusterGCN [32]. However, certain edges, such as task-irrelevant edges, may negatively impact performance, particularly in heterophilic or noisy graphs. In such cases, a high-quality learned sparse subgraph performs better than the full graph, as validated in our experiments (§VI-B).

Initializing $\tilde{p}$ with prior. While $\tilde{p}$ can be directly used to sample sparse subgraphs, the resulting subgraph may be suboptimal for message passing due to missing bridge edges connecting low-degree node pairs. Thus, augmenting the sampler with p_{prior}, which favors such edges, results in a better quality sparse subgraph (Case I in Fig. 1). p_{prior} is defined as

$$p_{\text{prior}}(u, v) \triangleq \frac{1/d_u + 1/d_v}{\sum_{i,j\in\mathcal{E}}(1/d_i + 1/d_j)}, \quad (16)$$

where d_u, d_v are degrees of nodes u, v.

We control the emphasis of prior on the learned distribution with a parameter $\lambda \in [0, 1]$, resulting in the *augmented probability distribution*: $\tilde{p}_a(u, v) = \lambda\tilde{p}(u, v) + (1-\lambda)p_{\text{prior}}(u, v)$ (line 7). The impact of p_{prior} on SGS-GNN is discussed in the ablation studies (§VI-G).

Conditional updates to EdgePE. Since backpropagation is computationally expensive, we only update EdgePE when the training F1-score from the learned sparse subgraph exceeds the baseline subgraph from p_{prior}. Thus, lines 5-13 in Alg. 2 are replaced by Alg. 3. This also helps our learned distribution to adjust so that the trained model performs no worse than fixed distribution sparsifiers.

During **inference**, we use the learned probability distribution from EdgePE, sample an ensemble of sparse subgraphs,

Algorithm 3 Training epoch with conditional update

1: **for** each graph partition $\mathcal{E}_i \in \mathcal{G}_{parts}$ **do**
2: $\tilde{p}, \boldsymbol{w} \leftarrow \texttt{EdgePE}_\phi(\mathcal{E}_i, p^i_{\text{prior}}, \boldsymbol{X}_i, \text{hops})$
3: $\tilde{\mathcal{E}}, \tilde{\boldsymbol{w}} \leftarrow$ Sample a subgraph using $\tilde{p}$
4: $\mathcal{E}_{\text{prior}} \leftarrow$ Sample a subgraph using p_{prior}
5: **if** Evaluate($\texttt{GNN}_\theta(\tilde{\mathcal{E}}, \tilde{\boldsymbol{w}})$) $\geq$ Evaluate($\texttt{GNN}_\theta(\mathcal{E}_{\text{prior}})$) **then**
6: Backpropagate through $\mathcal{L}$ to update $\texttt{EdgePE}_\phi$, $\texttt{GNN}_\theta$.
7: **else**
8: Backpropagate through $\mathcal{L}_{\text{CE}}$ to update $\texttt{GNN}_\theta$.
9: **end if**
10: **end for**

and mean-aggregate their representations to produce a final prediction on a test node.

Computational Complexity. Suppose the number of hidden dimensions $H \approx F$, where F is the dimension of the node features. The cost of an L-layer GCN is $\mathcal{O}(L(|\mathcal{E}| \cdot H + |\mathcal{V}| \cdot H^2))$ [32]. The cost of Alg 1 is $\mathcal{O}(L(|\mathcal{E}_{\text{sp}}| \cdot H + |\mathcal{V}| \cdot H^2) + |\mathcal{E}| \cdot H^2)$, since computing node-embedding (line 4, Alg. 1) using sparse graph $\mathcal{E}_{\text{sp}}$ costs $\mathcal{O}(L(|\mathcal{E}_{\text{sp}}| \cdot H + |\mathcal{V}| \cdot H^2))$, and edge weight computation using `MLP` (line 5, Alg. 1) costs $\mathcal{O}(|\mathcal{E}| \cdot H^2)$. With an L-layer GCN used as a downstream GNN acting on the sparse subgraph $\tilde{\mathcal{E}}$, the downstream GNN costs $\mathcal{O}(L(|\tilde{\mathcal{E}}| \cdot H + |\mathcal{V}| \cdot H^2)$. Since, $|\mathcal{E}_{\text{sp}}| = |\tilde{\mathcal{E}}|$ the total complexity of `SGS-GNN` (Alg. 2) is $\mathcal{O}(L(|\tilde{\mathcal{E}}| \cdot H + |\mathcal{V}| \cdot H^2) + |\mathcal{E}| \cdot H^2)$.

Space complexity. Let a graph be partitioned into n subgraphs with nearly equal sizes. The memory requirement for `SGS-GNN` with L-layer GCN is $\mathcal{O}\left(\frac{|\mathcal{E}|+|\mathcal{V}| \cdot H}{n} + L \cdot H^2\right)$.

V. Theory

In this section, we theoretically analyze `SGS-GNN`. The proofs and details of the analysis are in the extended paper.

A. Theoretical analysis I.

Let $\mathcal{E}^*$ and $\tilde{\mathcal{E}}$ be the ordered collection of edges sampled by the idealized learning ORACLE according to the true distribution p^* and by `SGS-GNN` according to learned probability $\tilde{p}$, respectively. For analytical purposes, let us assume that the algorithm samples k edges with replacement. We have the following theorem that lower-bounds the number of common edges between sampled subgraphs from `SGS-GNN` and the idealized learning ORACLE. Note that this ORACLE is a learning algorithm that knows the optimal edge sampling distribution. As the true distribution is unknown, such an ORACLE is impractical to build. Thus, it remains a hypothetical construct used to argue for the approximation quality.

Theorem V.1 (Lower-bound)**.** *The expected #edges sampled by both `SGS-GNN` and idealized learning ORACLE satisfies*

$$\mathbb{E}[|\mathcal{E}^* \cap \tilde{\mathcal{E}}|] \geq k \sum_{j=1}^{|\mathcal{E}|} \frac{(p^*_j + \tilde{p}_j - \epsilon)^2}{4}, \tag{17}$$

where $k = \lfloor q|\mathcal{E}|/100 \rfloor$ with $0 \leq q \leq 100$ as a user-specified parameter and $\epsilon \in [0, 1]$ is the error.

The implications are: (I) Let the true distribution be uniform. In the best-case scenario $\epsilon \to 0$ and $\tilde{p} = p^* = \frac{1}{|\mathcal{E}|}$. Then there are at least $\frac{k}{|\mathcal{E}|}$ common edges between $\tilde{\mathcal{G}}$ and $\mathcal{G}^*$. Since $k << |\mathcal{E}|$, this specific scenario suggests that the learned sparse subgraph may not overlap much with the true one even after we have learned the true distribution. When the true distribution is uniform, every subgraph from $\mathcal{G}_q$ is a global minimizer of the task-specific loss $\mathcal{L}_{CE}$. Otherwise, the learning ORACLE would have put more mass on certain edges, and the distribution p^* would not have been uniform. As individual subgraphs are indistinguishable in terms of performance, this case defeats the purpose of supervised sparsification. (II) Let the true distribution be one-hot. In other words, suppose $\tilde{p}_{ij} = p^*_{ij} = \delta_{ij}$, where δ_{ij} is the *Kronecker-delta*. In this case, as $\epsilon \to 0$, the lower bound reduces to

$$\mathbb{E}[|\mathcal{E}^* \cap \tilde{\mathcal{E}}|] \geq k \sum_{j=1}^{|\mathcal{E}|} (\tilde{p}_j)^2 = k.$$

This identity suggests that the sampled edges are expected to completely overlap with the true sparse subgraph.

For strong heterophilic graphs, the true distribution is less likely to be uniform, becaause a uniform edge sample would retain a similar node homophily as in the input graph, and such a subgraph would not be able to minimize $\mathcal{L}_{CE}$ [24]. Thus, it is important for the learned probability distribution to approximate p^* so that the sampled subgraph is close enough to the true one. We have analyzed $\tilde{\mathcal{G}}$ generated by `SGS-GNN` on a synthetic graph in our extended paper.

B. Theoretical analysis II.

We consider vanilla GCN as a downstream GNN to examine how the sparse subgraph, $\tilde{\mathcal{G}}$ from `SGS-GNN`, affects node embeddings compared to the ideal subgraph $\mathcal{G}^*$ from a learning ORACLE. Suppose an L-layer GCN produces embeddings $\tilde{\boldsymbol{H}}^{(L)}$ and $\boldsymbol{H}^{*(L)}$ when taking $\tilde{\mathcal{G}}$ and $\mathcal{G}^*$ as input, respectively. Can we upper-bound the difference in the downstream node encodings $\mathbb{E}[\|\tilde{\boldsymbol{H}}^{(L)} - \boldsymbol{H}^{*(L)}\|_2]$?

To that end, we assume $\forall l \in L$, $\|\boldsymbol{W}\|_2 \leq \alpha < 1$ where α is a constant. This is reasonable since each $\boldsymbol{W}^{(l)}$ is typically controlled during training using regularization techniques, e.g., weight decay. As input features in $\boldsymbol{X}$ are bounded, we also assume that there exists a constant β such that $\forall l > 0$, $\|\boldsymbol{H}\|_2^{(l)} \leq \beta$. We assume that σ is *Lipschitz continuous* with *Lipschitz constant* L_σ. We assume `ReLU` activation to simplify our analysis since `ReLU` has a Lipschitz constant $L_\sigma = 1$. Under these assumptions, we have the following theorem:

Theorem V.2 (Error in GCN encodings)**.** *For sufficiently deep L-layer GCN, the error in node embeddings*

$$\mathbb{E}[\lim_{L \to \infty} \|\tilde{\boldsymbol{H}}^{(L)} - \boldsymbol{H}^{*(L)}\|_2] < \frac{\beta}{1-\alpha} \sqrt{2k(1 - \sum_{j=1}^{|\mathcal{E}|} \frac{(p^*_j + \tilde{p}_j - \epsilon)^2}{4})}.$$

VI. Experiments

We experimented with 21 heterophilic and 12 homophilic benchmark datasets of varying sizes and homophily on node classification tasks (Table I). All experiments are carried out 10 times on a 24GB NVIDIA A10 Tensor Core GPU with 500

TABLE I

DATASET STATISTICS. HETEROPHILIC-LIGHT RED, HOMOPHILIC-LIGHT BLUE. d-AVERAGE DEGREE, C-#CLASSES, AND F-FEATURE DIMENSION.

DATASET	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	d	$\mathcal{H}_{\text{adj}}$	C	F
CORNELL	183	557	3.04	-0.42	5	1.70K
TEXAS	183	574	3.14	-0.26	5	1.70K
WISCONSIN	251	916	3.65	-0.20	5	1.70K
REED98	962	37.62K	39.11	-0.10	3	1.00K
AMHERST41	2.24K	181.91K	81.39	-0.07	3	1.19K
PENN94	41.55K	2.72M	65.56	-0.06	2	4.81K
ROMAN-EMPIRE	22.66K	65.85K	2.91	-0.05	18	300
CORNELL5	18.66K	1.58M	84.76	-0.04	3	4.74K
SQUIRREL	5.20K	396.85K	76.30	-0.01	5	2.35K
JOHNSHOPKINS55	5.18K	373.17K	72.04	0.00	3	2.41K
ACTOR	7.60K	53.41K	7.03	0.01	5	932
MINESWEEPER	10.00K	78.80K	7.88	0.01	2	7
QUESTIONS	48.92K	307.08K	6.28	0.02	2	301
CHAMELEON	2.28K	62.79K	27.58	0.03	5	2.58K
TOLOKERS	11.76K	1.04M	88.28	0.09	2	10
AMAZON-RATINGS	24.49K	186.10K	7.60	0.14	5	556
GENIUS	421.96K	1.85M	4.37	0.17	2	12
POKEC	1.63M	44.60M	27.32	0.42	3	65
ARXIV-YEAR	169.34K	2.32M	13.67	0.26	5	128
SNAP-PATENTS	2.92M	27.95M	9.56	0.21	5	269
OGBN-PROTEINS	132.53K	79.12M	597.00	0.05	94	8
CORA	19.79K	126.84K	6.41	0.56	70	8.71K
DBLP	17.72K	105.73K	5.97	0.68	4	1.64K
COMPUTERS	13.75K	491.72K	35.76	0.68	10	767
PUBMED	19.72K	88.65K	4.50	0.69	3	500
CORA_ML	2.99K	16.32K	5.45	0.75	7	2.88K
SMALLCORA	2.71K	10.56K	3.90	0.77	7	1.43K
CS	18.33K	163.79K	8.93	0.78	15	6.80K
PHOTO	7.65K	238.16K	31.13	0.79	8	745
PHYSICS	34.49K	495.92K	14.38	0.87	5	8.42K
CITESEER	4.23K	10.67K	2.52	0.94	6	602
WIKI	11.70K	431.73K	36.90	0.58	10	300
REDDIT	232.97K	114.62M	491.99	0.74	41	602

GB internal memory, and with a data split of $20\%/40\%/40\%$ (train/validation/test). For baseline models, we follow the settings set by respective authors. We use 2 message passing layers for SGS−GNN and a hidden layer dimension of $H = 256$ for both $\texttt{EdgePE}_\phi$ and $\texttt{GNN}_\theta$. We set a dropout rate of 0.2 and use the Adam optimizer with a learning rate of 0.001. We trained all models for a maximum epoch of 500 with early stopping. The edge batch size is $500K$, and the number of partitions for METIS is $n = \lceil |\mathcal{E}|/500K \rceil$. The percentage of edge sample is set to $q = 20\%$ following DropEdge [14]. We take 10 samples of sparse subgraphs during inference. The model with the best validation F1-score is selected for testing. Our source codes are anonymously provided on Github[1].

Baselines. We use ClusterGCN [32] to evaluate performance on the original large graphs. DropEdge [14], and GraphSAINT (GSAINT-E) [13] are used as fixed distribution samplers. For Mixture of Graph (MoG) [3], we use 3 experts and their recommended settings. We adjust the neighborhood sample size of NeuralSparse [9] to match the sparsity of ours for a fair comparison. We included SparseGAT [11] as another supervised method for generating sparse graphs. Additionally, we compare our approach with heterophilic GNNs (H2GCN [17], ASGC [33]), sparsifiers based on the lottery ticket hypothesis (UGT-GCN [22]), and others (DSpar [34], CGP-GCN [35]).

Key Questions. In this section, we investigate: (i) How much memory does SGS−GNN reduce (§VI-A) over related sparsifiers? (ii) How does SGS−GNN perform compared to baselines that employ a fixed distribution sampling over edges? (§VI-B) (iii) How does SGS−GNN fare against supervised and unsupervised sparsifiers, and other related baselines? (§VI-C) (iv) By how much does SGS−GNN reduce runtime (§VI-D) and converge faster (§VI-E) over related work? (v) How accurately does SGS−GNN perform with different levels of sparsity and homophily in the underlying graph? (§VI-F) (vi) What is the impact of each component of SGS−GNN on its prediction quality? (Ablation studies §VI-G)

[1] https://github.com/anonymousauthors001/SGS-GNN/

A. Memory

Table VI-A shows that SGS−GNN consumes significantly less memory than three related sparsification methods across 8 representative datasets. This is because it avoids using the full graph during the forward pass of embedding computation in EdgePE and GNN. SGS−GNN ensures gradient flow through sampled subgraphs and training edges in the regularizers. In contrast, other supervised methods maintain gradient flow through a mask that is the same size as the original graph, which does not help reduce memory. By partitioning graphs, SGS−GNN solves problems that other methods cannot handle due to memory limitations. SGS−GNN takes up to $4\times$ less memory compared to NeuralSparse, SparseGAT, and MOG.

TABLE II

GPU MEMORY CONSUMPTION (IN MB) OF SUPERVISED METHODS. # INDICATES THE NUMBER OF GRAPH PARTITIONS FROM METIS. *OOM* - OUT OF MEMORY. S.GAT = SPARSEGAT, NSPARSE = NEURALSPARSE.

DATA.	NSPARSE	S.GAT	MOG	SGS−GNN	#
AMHE.	2,479	3,197	7,617	**1,319**	1
AMAZ.	3,941	2,749	4,609	**1,471**	1
TOLO.	5,735	8,681	4,341	**5,639**	1
JOHN.	8,319	6,237	OOM	**2,333**	1
CORN.	OOM	19,243	OOM	**8,725**	1
ARXI.	22,590	22,535	16,728	**10,341**	1
ARXI.	22,590	22,535	16,728	**2,958**	10
WIKI	3,709	4,051	5,641	**1,537**	1
REDD.	OOM	OOM	OOM	**15,355**	32
REDD.	OOM	OOM	OOM	**3,041**	256

B. SGS−GNN vs. fixed distr. sparsifiers

We compare SGS−GNN with fixed edge distribution sparsifiers like *Random* from DropEdge, *Edge* sampler from GraphSAINT, and *Effective resistance (ER)*. Table III presents F1-scores, with the last row summarizing overall performance. The *Org. Graph* represents the original dense graph's performance computed using ClusterGCN. Our learnable sampler EdgePE significantly outperforms fixed distribution samplers and original dense graphs on heterophilic datasets. On homophilic graphs, we observe a smaller margin of improvement, but SGS−GNN still outperforms other baselines.

C. SGS−GNN vs. other Sparsification baselines

Table IV compares SGS−GNN with related sparsification-based GNNs. We use GCN as the GNN module in our SGS−GNN. SGS−GNN significantly outperforms competing methods with a geometric mean improvement of $4-5\%$ with only 20% of edges. Under similar settings, SGS−GNN significantly outperforms in heterophilic graphs and remains competitive in homophilic graphs.

TABLE III

MEAN F1-SCORES (IN %) ± STD. DEV. OF VARIOUS FIXED DISTRIBUTION SAMPLERS USING 20% EDGES. **BOLD** INDICATES BEST-PERFORMING SAMPLER EXCLUDING *Org. graph*. **GM.** - GEOMETRIC MEAN.

DATA.	ORG. GRAPH	RANDOM	EDGE	ER	SGS-GNN
CORN.	43.78 ± 4.32	49.19 ± 4.65	46.49 ± 2.65	43.78 ± 3.97	**74.59 ± 1.32**
TEXA.	61.62 ± 1.08	55.14 ± 5.01	69.19 ± 2.76	61.08 ± 2.76	**76.22 ± 2.02**
WISC.	51.76 ± 5.49	61.96 ± 4.40	66.27 ± 2.29	58.82 ± 2.15	**76.08 ± 3.14**
REED.	61.35 ± 0.84	54.92 ± 2.64	54.51 ± 1.98	59.69 ± 2.03	**64.15 ± 2.28**
AMHE.	61.83 ± 0.30	57.49 ± 0.81	57.40 ± 0.58	50.60 ± 1.14	**72.75 ± 0.59**
PENN.	73.23 ± 0.05	68.52 ± 0.34	68.35 ± 0.36	70.74 ± 0.39	**75.65 ± 0.41**
ROMA.	44.25 ± 0.16	43.08 ± 0.61	42.91 ± 0.63	58.18 ± 1.25	**64.69 ± 0.12**
CORN.	65.13 ± 0.31	63.36 ± 0.50	63.47 ± 0.46	63.89 ± 0.35	**69.15 ± 0.33**
SQUI.	48.38 ± 0.65	42.40 ± 0.96	42.44 ± 1.10	43.86 ± 0.42	**52.35 ± 0.35**
JOHN.	68.71 ± 0.28	63.40 ± 0.53	62.80 ± 0.74	62.14 ± 2.51	**73.80 ± 0.33**
ACTO.	28.42 ± 0.23	32.37 ± 0.78	30.53 ± 0.59	32.03 ± 0.27	**33.88 ± 0.42**
MINE.	79.56 ± 0.03	79.73 ± 0.12	79.84 ± 0.06	**80.02 ± 0.03**	80.00 ± 0.00
QUES.	97.05 ± 0.01	97.07 ± 0.03	**97.08 ± 0.01**	97.02 ± 0.00	97.05 ± 0.01
CHAM.	64.43 ± 0.43	57.98 ± 1.39	57.11 ± 1.22	59.78 ± 0.85	**62.37 ± 0.98**
TOLO.	79.03 ± 0.15	78.59 ± 0.16	78.59 ± 0.21	78.10 ± 0.06	**79.98 ± 0.17**
AMAZ.	46.72 ± 0.20	45.70 ± 0.21	45.75 ± 0.35	44.39 ± 0.12	**50.15 ± 0.34**
GENI.	80.80 ± 0.02	81.99 ± 0.09	81.60 ± 0.03	82.25 ± 0.86	**82.59 ± 0.00**
POKE.	62.05 ± 0.37	60.30 ± 0.27	60.17 ± 0.17	58.76 ± 0.59	**60.49 ± 0.10**
ARXI.	39.05 ± 0.08	36.96 ± 0.01	37.06 ± 0.04	36.62 ± 0.33	**38.42 ± 0.10**
SNAP.	35.38 ± 0.15	34.57 ± 0.08	34.48 ± 0.16	33.13 ± 0.37	**35.41 ± 0.10**
OGBN.	93.15 ± 0.00	93.15 ± 0.00	93.15 ± 0.00	93.15 ± 0.00	93.15 ± 0.00
GM.	58.42	57.25	57.63	57.68	64.79
CORA.	67.29 ± 0.51	61.20 ± 7.76	57.63 ± 14.73	**66.90 ± 0.17**	65.58 ± 0.69
DBLP.	83.92 ± 0.04	81.00 ± 0.27	81.19 ± 0.33	**81.81 ± 0.10**	80.37 ± 0.16
COMP.	90.19 ± 0.18	90.34 ± 0.29	90.37 ± 0.25	89.87 ± 0.96	**90.97 ± 0.31**
PUBM.	86.73 ± 0.07	87.58 ± 0.22	87.62 ± 0.14	**87.70 ± 0.11**	87.52 ± 0.15
CORA.	86.29 ± 0.51	85.39 ± 0.35	85.29 ± 0.60	**85.63 ± 0.51**	83.99 ± 0.53
SMAL.	80.28 ± 0.37	75.82 ± 0.54	76.44 ± 1.21	75.90 ± 1.25	**76.94 ± 0.76**
CS.	92.79 ± 0.10	94.07 ± 0.14	94.09 ± 0.09	93.77 ± 0.17	**94.25 ± 0.15**
PHOT.	92.41 ± 2.01	93.54 ± 0.14	93.63 ± 0.25	93.42 ± 0.10	**93.99 ± 0.25**
PHYS.	96.08 ± 0.03	96.20 ± 0.07	96.23 ± 0.12	96.22 ± 0.07	**96.27 ± 0.09**
CITE.	91.44 ± 0.29	86.38 ± 0.26	86.75 ± 0.22	86.24 ± 0.26	**86.78 ± 0.28**
WIKI.	80.07 ± 0.21	80.10 ± 0.13	80.19 ± 0.16	80.32 ± 0.13	**81.49 ± 0.31**
REDD.	91.43 ± 0.07	91.39 ± 0.06	91.35 ± 0.08	91.00 ± 0.06	**91.45 ± 0.06**
GM.	86.22	84.68	84.37	85.33	85.36

TABLE IV

MEAN F1-SCORES (IN %) ± STD. DEV. OF BASELINE SPARSIFIERS USING 20% EDGES. **BOLD** - BEST-PERFORMING METHOD. OOM - OUT OF MEMORY. **GM.** - GEOMETRIC MEAN. THE GEOMETRIC MEAN IS COMPUTED ACROSS THAT DATASET WHERE ALL METHODS HAVE RESULTS.

DATA.	GSAINT-E	DROPEDGE	MOG	SPARSEGAT	NEURALSP.	SGS-GNN
CORN.	46.49 ± 2.65	43.24 ± 2.20	42.16 ± 3.08	51.35 ± 0.10	72.43 ± 6.48	**74.59 ± 1.32**
TEXA.	69.19 ± 2.76	54.95 ± 3.37	57.30 ± 4.83	66.66 ± 1.27	**84.44 ± 1.53**	76.22 ± 2.02
WISC.	66.27 ± 2.29	48.36 ± 0.92	53.33 ± 1.64	56.86 ± 0.00	52.83 ± 47.00	**76.08 ± 3.14**
REED.	54.51 ± 1.98	60.03 ± 0.05	55.75 ± 2.61	55.69 ± 0.25	58.54 ± 1.60	**64.15 ± 2.28**
AMHE.	57.40 ± 0.58	59.00 ± 18.80	56.78 ± 2.05	49.00 ± 0.20	56.85 ± 75.00	**72.75 ± 0.59**
PENN.	68.35 ± 0.36	65.87 ± 0.30	OOM	65.00 ± 0.10	OOM	**75.65 ± 0.41**
ROMA.	42.91 ± 0.63	46.00 ± 1.20	39.27 ± 0.60	41.18 ± 0.07	44.91 ± 5.79	**64.69 ± 0.12**
CORN.	63.47 ± 0.46	61.40 ± 1.45	OOM	53.77 ± 0.39	OOM	**69.15 ± 0.33**
SQUI.	42.44 ± 1.10	48.60 ± 0.00	27.67 ± 0.51	29.50 ± 0.00	38.24 ± 0.00	**52.35 ± 0.35**
JOHN.	62.80 ± 0.74	64.14 ± 1.75	OOM	57.57 ± 0.29	57.56 ± 1.06	**73.80 ± 0.33**
ACTO.	30.53 ± 0.59	34.64 ± 1.40	27.74 ± 0.96	25.05 ± 0.60	27.85 ± 0.19	**33.88 ± 0.42**
MINE.	79.84 ± 0.06	**80.00 ± 0.00**	**80.00 ± 0.00**	**80.00 ± 0.00**	**80.00 ± 0.10**	**80.00 ± 0.00**
QUES.	**97.08 ± 0.01**	97.00 ± 0.01	97.04 ± 0.01	**97.08 ± 0.01**	97.02 ± 0.01	97.05 ± 0.01
CHAM.	57.11 ± 1.22	50.60 ± 0.04	53.25 ± 0.63	60.60 ± 0.15	60.52 ± 0.78	**62.37 ± 0.98**
TOLO.	78.59 ± 0.21	78.40 ± 0.20	78.49 ± 0.28	78.20 ± 0.72	78.16 ± 0.00	**79.98 ± 0.17**
AMAZ.	45.75 ± 0.35	43.87 ± 0.67	41.18 ± 0.49	44.23 ± 0.05	47.05 ± 0.47	**50.15 ± 0.34**
GM.	56.44	55.04	51.15	53.04	58.25	65.99
CORA	57.63 ± 14.73	65.09 ± 0.44	**67.26 ± 1.11**	61.07 ± 0.39	56.68 ± 0.32	65.58 ± 0.69
DBLP	81.19 ± 0.33	87.20 ± 0.15	72.37 ± 0.63	**84.68 ± 1.00**	73.39 ± 0.67	80.37 ± 0.16
COMP.	90.37 ± 0.25	60.65 ± 4.66	OOM	88.91 ± 0.00	75.32 ± 4.11	**90.97 ± 0.31**
PUBM.	**87.62 ± 0.14**	86.00 ± 1.21	83.84 ± 0.58	75.30 ± 0.35	73.97 ± 0.40	87.52 ± 0.15
CORA.	**85.29 ± 0.60**	84.80 ± 0.20	OOM	80.60 ± 0.40	79.30 ± 0.87	83.99 ± 0.53
SMAL.	76.44 ± 1.21	76.47 ± 0.31	78.43 ± 0.73	75.70 ± 0.43	75.79 ± 9.00	**76.94 ± 0.76**
CS	94.09 ± 0.09	93.30 ± 0.32	72.88 ± 0.32	92.35 ± 0.06	94.36 ± 0.40	**94.25 ± 0.15**
PHOT.	93.63 ± 0.25	80.47 ± 59.10	83.84 ± 0.58	**95.30 ± 0.02**	94.16 ± 0.25	93.99 ± 0.25
PHYS.	96.23 ± 0.12	97.28 ± 0.70	OOM	95.96 ± 0.06	96.38 ± 0.04	**96.27 ± 0.09**
CITE.	86.75 ± 0.22	77.40 ± 0.10	78.43 ± 0.73	58.75 ± 0.10	65.40 ± 1.20	**86.78 ± 0.28**
WIKI	80.19 ± 0.16	80.19 ± 0.16	72.88 ± 0.32	79.26 ± 0.11	73.03 ± 1.10	**81.49 ± 0.31**
GM.	81.36	80.36	76.04	76.78	74.89	82.87

Table V compares the performance of SGS−GNN against other related baselines: two heterophilic GNNs: H2GCN [17] and SGC [33], one lottery ticket-based method: UGT-GCN [22], and two sparsifiers: CGP-GCN [35] and DSPar [34]. Our results indicate that SGS−GNN outperforms these sparsifiers while remaining competitive with heterophilic GNNs (H2GCN and SGC) that use full graphs, even if in this experiment SGS−GNN used homophilic GNN.

TABLE V

H2GCN, ASGC - HETEROPHILIC GNNS, CGP-GCN - UNSUPERVISED SPARSIFIER, UGT-GCN - LOTTERY TICKET. WE HIGHLIGHT THE BEST PERFORMANCE BETWEEN SPARSIFIERS. RED - NOT SPARSIFIERS, BLUE - SPARSIFIERS. OOM - OUT OF MEMORY, (-) -COMPILATION ERROR.

DATA.	H2GCN	ASGC	CGP-GCN	UGT-GCN	DSPAR-GCN	SGS-GCN
TOLO.	79.17±0.10	79.01±0.00	78.85±0.06	78.16±0.31	78.37±0.02	**79.98±0.17**
ROMA.	78.21±0.19	64.37±0.30	53.68±0.15	-	57.03±0.23	**64.69±0.12**
JOHN.	63.00±0.33	75.77±0.03	69.43±0.30	66.99±1.01	61.37±1.23	**73.80±0.33**
AMHE.	63.31±0.37	78.08±0.00	65.62±1.30	62.86±2.35	58.88±1.61	**72.75±0.59**
CORN.	68.00±0.26	-	67.68±0.21	63.96±0.06	63.31±0.45	**69.15±0.33**
GENI.	82.38±0.34	-	82.49±0.18	OOM	82.11±0.01	**82.59±0.00**
POKE.	OOM	OOM	OOM	OOM	**60.92±0.01**	60.49±0.10
ARXI.	40.74±0.10	-	**40.60±0.02**	OOM	37.58±0.09	38.42±0.10
PHOT.	94.67±0.14	94.31±0.00	93.67±0.09	92.25±0.07	89.15±0.25	**93.99±0.25**
CS	94.70±0.06	94.10±0.00	-	93.76±0.01	91.60±0.02	**94.25±0.15**
REDD.	OOM	OOM	OOM	OOM	74.49±0.07	**91.45±0.06**

Moreover, SGS−GNN can handle large graphs (e.g., Pokec, Reddit) while CGP-GCN, UGT-GCN, SparseGAT, MoG, and NeuralSparse ran out of memory in our experiments.

D. Runtime

Table VI compares the training times per epoch of SGS−GNN with other GNN-based sparsifier baselines. Under similar conditions, SGS−GNN is more efficient than NeuralSparse, MoG, and competitive with SparseGAT. Since SparseGAT is an implicit sparsifier, unlike SGS−GNN, it is not memory efficient and cannot handle large graphs. Unsupervised sparsifiers such as DropEdge and GraphSAINT are faster but not always as accurate as SGS−GNN.

TABLE VI

MEAN TRAINING TIME PER EPOCH (IN MILLISECONDS). RED - UNSUPERVISED SPARSIFIERS, BLUE - SUPERVISED SPASRIFIERS.

DATA.	CLU.GCN	GSAINT	D.EDGE	MOG	S.GAT	NEURALSP.	SGS-GNN
AMAZ.	6.8	6.2	17.0	110.0	15.0	50.0	18.0
JOHN.	7.1	6.1	21.0	OOM	10.0	120.0	24.0
AMHE.	6.2	5.8	10.0	OOM	5.3	37.0	16.0
CS	9.5	8.9	15.0	OOM	100.0	150.0	22.0
QUES.	8.2	7.2	29.0	130.0	24.0	120.0	26.0

Table VII compares the runtime of SGS−GNN with and without conditional updates for large-scale graphs (with $|\mathcal{E}| \geq 1M$). The results indicate that conditional updates are similar to our standard training algorithm in terms of computational efficiency while providing improvements in F1-score under identical conditions. Fewer updates of EdgePE compensate for the additional computational costs of evaluation with prior.

E. Convergence

To compare the sparsifiers in terms of convergence, we terminate training when the std. dev of loss in five consecutive epochs is $\leq 10^{-3}$. Fig. 3 shows the bar plot of the number of epochs required for the methods to converge. SGS−GNN requires fewer iterations than other fixed distribution sparsifiers highlighting the benefit of learning the distribution.

TABLE VII

COMPARISON OF RUNTIME OF SGS−GNN WITH AND WITHOUT CONDITIONAL UPDATES ON LARGE-SCALE GRAPHS (WITH $|\mathcal{E}| \geq 1M$). THE TERMS EDGEPE/GNN REPRESENT THE PROPORTION OF TIMES THE EDGEPE MODULE IS UPDATED RELATIVE TO THE GNN.

DATA.	RUNTIME (S)		F1-SCORE		#EDGEPE/ #GNN
	W/O. COND	W. COND	W/O. COND	W. COND	
TOLO.	0.17	**0.16**	78.12 ± 0.13	$\mathbf{78.13 \pm 0.17}$	0.42
GENI.	**0.38**	0.48	79.92 ± 0.08	$\mathbf{80.07 \pm 0.11}$	0.43
POKEC	6.79	**6.48**	62.05 ± 0.33	$\mathbf{62.20 \pm 0.10}$	0.75
ARXI.	**0.45**	0.45	$\mathbf{36.99 \pm 0.11}$	36.98 ± 0.13	0.23
SNAP.	**6.34**	7.12	34.86 ± 0.15	$\mathbf{34.95 \pm 0.16}$	0.84
REDD.	**8.08**	8.29	$\mathbf{91.45 \pm 0.07}$	91.43 ± 0.02	0.44

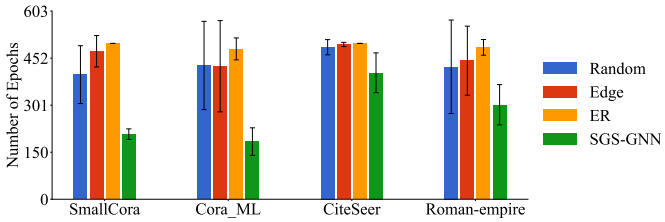

Fig. 3. Number of epochs required by SGS−GNN to converge compared to other samplers under the same settings.

F. Impact of Sparsity and Homophily on Accuracy

We analyzed the performance of SGS−GNN across varying homophily levels using synthetic and benchmark graphs in Fig. 4. Synthetic graphs were generated with nodes of homophily $\mathcal{H}_n$ and degree d by connecting $\lceil d\mathcal{H}_n \rceil$ edges neighbors of the same class, and $d - \lceil d\mathcal{H}_n \rceil$ edges randomly. Results in Fig. 4(a) show that high homophily yields good performance regardless of sparsity, while high sparsity benefits heterophily, with optimal performance seen at $30\% - 40\%$ edge retention for heterophily levels $0.3 - 0.4$. Furthermore, Fig. 4(b) indicates that while more edges improve accuracy on homophilic graphs, high sparsity can be advantageous on heterophilic graphs such as reed98 and amherst41, where we observe best performance at $\sim 20\%$ sparsity.

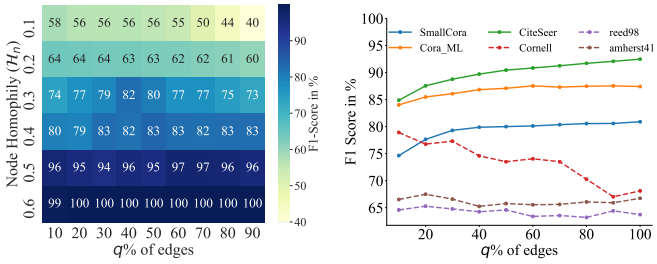

Fig. 4. (Left) Heatmap of F1 scores (in %) at different homophily and sparsity levels (q) in Cora synthetic graphs. (Right) F1-scores of SGS−GNN at different sparsity for homophilic (solid line) and heterophilic (dashed line) graphs.

Fig. 5 shows that the sampled subgraphs of SGS−GNN have higher *edge homophily* than those from the fixed distribution sparsifiers. This is expected due to our $\mathcal{L}_{\text{assor}}$ regularizer.

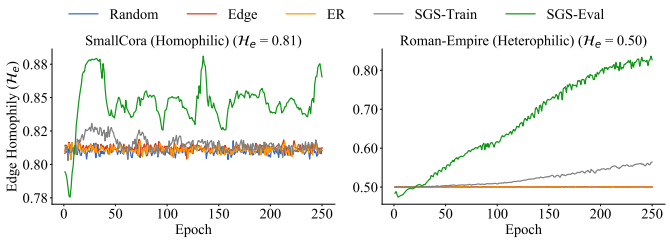

Fig. 5. Edge homophily of selected subgraphs from different fixed distribution samplers vs. subgraphs from training and evaluation phase of SGS−GNN.

G. Ablation Studies

a) Impact of components (I): Table VIII shows performance of SGS−GNN with various combinations of $\mathcal{L}_{\text{assor}}, \mathcal{L}_{cons}$, EdgePE, GNN, and Conditional Updates. **(I)** $\mathcal{L}_{\text{assor}}$**:** Cases 1, 2 show improvement in results when $\mathcal{L}_{\text{assor}}$ is used. **(II)** $\mathcal{L}_{\text{cons}}$**:** Cases 3, 4, 5 show L_{cons} improves results when GCN module is used in the GNN. **(III) Conditional updates:** Case 3 shows that conditional updates can benefit some graphs. **(IV) EdgePE:** In general, we found that the GCN layers for EdgePE encodings perform best (cases 4, 8). **(V) GNN:** Both GCN and GAT modules yielded overall the best results (Cases 4, 8).

TABLE VIII

EDGEPE, GNN, CONDITIONAL UPDATE AND REGULARIZERS.

C.	$L_{as.}$	$L_{cn.}$	EDGEPE	GNN	COND.	SMALLCORA	CORAFULL	JOHNS.
1	N	N	MLP	GCN	N	73.80 ± 0.67	61.78 ± 0.20	66.12 ± 1.38
2	Y	N	MLP	GCN	N	74.88 ± 0.15	63.99 ± 0.24	66.18 ± 1.05
3	Y	N	GCN	GCN	Y	75.80 ± 0.77	65.66 ± 0.14	71.06 ± 0.32
4	Y	Y	GCN	GCN	Y	77.50 ± 0.62	$\mathbf{66.56 \pm 0.22}$	70.79 ± 0.18
5	Y	Y	GCN	GCN	N	75.88 ± 0.71	65.87 ± 0.17	69.83 ± 0.67
6	Y	N	GSAGE	GCN	Y	75.82 ± 0.44	63.70 ± 0.09	67.53 ± 0.80
7	Y	Y	GSAGE	GCN	Y	77.48 ± 0.61	65.12 ± 0.11	68.63 ± 0.66
8	Y	N	GCN	GAT	Y	$\mathbf{78.18 \pm 0.74}$	66.33 ± 0.20	$\mathbf{71.97 \pm 0.59}$

b) Impact of components (II): Table IX highlights the impact of p_{prior}, Normalization & Sampling schemes, and Ensembling on SGS−GNN. **(I) Prior** Cases 2-3 show that augmenting the learned probability distribution $\tilde{p}$ with prior p_{prior} can benefit some datasets. **(II) Normalization and Sampling:** Cases 3-5 show that each of the sampling techniques can improve results in certain datasets, and it isn't easy to nominate a single one as best. However, in our experiments, we opted for multinomial sampling with softmax temperature annealing for training to encourage exploration in early iterations. **(III) Ensemble subgraphs during inference:** Case 2 demonstrates that using multiple subgraphs for ensemble prediction yields better results than a single subgraph (Case 1).

c) SGS−GNN with and without Graph Partitioning: Partitioning large graphs is critical to solve large problems, avoiding the memory bottleneck of supervised classification methods. However, partitioning increases the training time. Additionally, the batch processing of subgraphs in the partitions prevents the method from being stuck in local minima, thereby improving accuracy (see Table X). On small graphs, we found negligible effects of partitioning on SGS-GNN's prediction quality.

TABLE IX
PRIOR, NORMALIZATION, SAMPLING, AND ENSEMBLE INFERENCE.

C.	*P.*	NORM.	SAMPL.	ENS.	SMALLCORA	JOHNS.	ROMAN.
1	N	SUM	MULT	N	69.30 ± 1.20	63.86 ± 0.58	63.27 ± 0.31
2	N	SUM	MULT	Y	72.84 ± 0.91	65.14 ± 1.14	$\mathbf{64.31 \pm 0.13}$
3	Y	SUM	MULT	Y	75.54 ± 0.41	$\mathbf{72.68 \pm 0.51}$	62.88 ± 0.19
4	Y	SOFT	MULT	Y	75.44 ± 0.51	$\mathbf{72.97 \pm 0.20}$	62.98 ± 0.16
5	Y	GUMBEL	TOPK	Y	$\mathbf{76.24 \pm 0.43}$	71.83 ± 1.00	63.00 ± 0.11

P: PRIOR, **SUM:** SUM-NORMALIZATION, **SOFT.:** *SoftMax* WITH TEMPERATURE ANNEALING. **MULT:** *Multinomial*, **GUMBEL:** *Gumbel-Softmax* WITH *TopK*, **ENS.** ENSEMBLE INFERENCE.

TABLE X
SGS−GNN TRAINING TIME (T) WITH AND WITHOUT METIS (#-PARTS).

DATASET	WITHOUT METIS		WITH METIS		#
	T/ITR. (#ITR.)	F1	T/ITR. (#ITR.)	F1	
SMALLCORA	0.01 (192)	**76.94±0.76**	0.08 (120)	76.27±0.29	5
ARXIV-YEAR	0.16 (52)	35.42±0.04	1.34 (109)	**37.17±0.51**	5

VII. CONCLUSION

We proposed SGS−GNN, a supervised graph sparsifier that produces a sparse subgraph with user-prescribed sparsity facilitating GNN computation on large-scale graphs. We provided a theoretical analysis of SGS−GNN in terms of the quality of embedding it produces compared to an idealized oracle sparsifier. Finally, we empirically validated the effectiveness, efficiency, and convergence of SGS−GNN over several baselines across homophilic and heterophilic graphs of various sizes. In the future, we plan to explore the robustness of our sparsifier in the presence of noise.

REFERENCES

[1] L. Wu, P. Cui, J. Pei, and L. Zhao, *Graph Neural Networks: Foundations, Frontiers, and Applications*. Singapore: Springer Singapore, 2022.

[2] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph Neural Networks: A review of methods and applications," *AI open*, vol. 1, pp. 57–81, 2020.

[3] G. Zhang, X. Sun, Y. Yue, C. Jiang, K. Wang, T. Chen, and S. Pan, "Graph Sparsification via Mixture of Graphs," *arXiv:2405.14260*, 2024.

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

[5] Y. Chen, H. Ye, S. Vedula, A. Bronstein, R. Dreslinski, T. Mudge, and N. Talati, "Demystifying graph sparsification algorithms in graph properties preservation," *Proceedings of the VLDB Endowment*, vol. 17, no. 3, pp. 427–440, 2023.

[6] M. Hashemi, S. Gong, J. Ni, W. Fan, B. A. Prakash, and W. Jin, "A comprehensive survey on graph reduction: sparsification, coarsening, and condensation," *arXiv:2402.03358*, 2024.

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

[8] F. F. Dragan, F. V. Fomin, and P. A. Golovach, "Spanners in sparse graphs," *Journal of Computer and System Sciences*, vol. 77, no. 6, pp. 1108–1119, 2011.

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

[10] D. Luo, W. Cheng, W. Yu, B. Zong, J. Ni, H. Chen, and X. Zhang, "Learning to drop: Robust Graph Neural Network via topological denoising," in *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, 2021, pp. 779–787.

[11] Y. Ye and S. Ji, "Sparse graph attention networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 1, pp. 905–916, 2021.

[12] D. Kim and A. Oh, "How to find your friendly neighborhood: Graph attention design with self-supervision," *arXiv:2204.04879*, 2022.

[13] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "Graph-SAINT: Graph sampling based inductive learning method," in *International Conference on Learning Representations*, 2019.

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

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

[16] H. Pei, B. Wei, K. C.-C. Chang, Y. Lei, and B. Yang, "Geom-GCN: Geometric graph convolutional networks," *arXiv:2002.05287*, 2020.

[17] J. Zhu, Y. Yan, L. Zhao, M. Heimann, L. Akoglu, and D. Koutra, "Beyond homophily in graph neural networks: Current limitations and effective designs," *Advances in Neural Information Processing Systems*, vol. 33, pp. 7793–7804, 2020.

[18] O. Platonov, D. Kuznedelev, A. Babenko, and L. Prokhorenkova, "Characterizing graph datasets for node classification: Beyond homophily-heterophily dichotomy," *arXiv:2209.06177*, 2022.

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

[20] R. S. Srinivasa, C. Xiao, L. Glass, J. Romberg, and J. Sun, "Fast graph attention networks using effective resistance based graph sparsification," *arXiv:2006.08796*, 2020.

[21] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, 2017, pp. 1024–1034.

[22] T. Chen, Y. Sui, X. Chen, A. Zhang, and Z. Wang, "A unified lottery ticket hypothesis for Graph Neural Networks," in *International conference on machine learning*. PMLR, 2021, pp. 1695–1706.

[23] X. Xu, N. Yuruk, Z. Feng, and T. A. Schweiger, "SCAN: A structural clustering algorithm for networks," in *Proceedings of the 13th ACM SIGKDD international conference on knowledge discovery and data mining*, 2007, pp. 824–833.

[24] S. S. Das, S. Ferdous, M. M. Halappanavar, E. Serra, and A. Pothen, "AGS-GNN: Attribute-guided sampling for graph neural networks," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2024, pp. 538–549.

[25] J. Li, T. Zhang, H. Tian, S. Jin, M. Fardad, and R. Zafarani, "SGCN: A graph sparsifier based on graph convolutional networks," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2020.

[26] S. Suresh, P. Li, C. Hao, and J. Neville, "Adversarial graph augmentation to improve graph contrastive learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 15 920–15 933, 2021.

[27] Z. Wang, Y. He, and B. Liu, "Probability passing for graph neural networks: Graph structure and representations joint learning," *arXiv:2407.10688*, 2024.

[28] H. Chen, Y. Xu, F. Huang, Z. Deng, W. Huang, S. Wang, P. He, and Z. Li, "Label-aware graph convolutional networks," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 1977–1980.

[29] L. Lovász, "Random walks on graphs," *Combinatorics, Paul erdos is eighty*, vol. 2, no. 1-46, p. 4, 1993.

[30] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with Gumbel-Softmax," *arXiv:1611.01144*, 2016.

[31] G. Karypis, "Metis: Unstructured graph partitioning and sparse matrix ordering system," *Technical report*, 1997.

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

[33] S. Chanpuriya and C. Musco, "Simplified graph convolution with heterophily," *Advances in neural information processing systems*, vol. 35, pp. 27 184–27 197, 2022.

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

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

[36] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.
[37] M. T. Augustine, "A survey on universal approximation theorems," *arXiv preprint arXiv:2407.12895*, 2024.
[38] R. Xie, *Distributionally Robust Optimization and its Applications in Power System Energy Storage Sizing*. Springer Nature, 2024.
[39] B. Rozemberczki, C. Allen, and R. Sarkar, "Multi-scale attributed node embedding," *Journal of Complex Networks*, vol. 9, no. 2, p. cnab014, 2021.
[40] D. Lim, F. Hohne, X. Li, S. L. Huang, V. Gupta, O. Bhalerao, and S. N. Lim, "Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods," *Advances in Neural Information Processing Systems*, vol. 34, pp. 20 887–20 902, 2021.
[41] O. Platonov, D. Kuznedelev, M. Diskin, A. Babenko, and L. Prokhorenkova, "A critical look at the evaluation of gnns under heterophily: Are we really making progress?" *arXiv:2302.11640*, 2023.
[42] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI magazine*, vol. 29, no. 3, pp. 93–93, 2008.
[43] C. L. Giles, K. D. Bollacker, and S. Lawrence, "Citeseer: An automatic citation indexing system," in *Proceedings of the third ACM Conference on Digital Libraries*, 1998, pp. 89–98.
[44] G. Namata, B. London, L. Getoor, B. Huang, and U. Edu, "Query-driven active surveying for collective classification," in *10th International Workshop on Mining and Learning with Graphs*, vol. 8, 2012, p. 1.
[45] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," *arXiv:1811.05868*, 2018.
[46] X. Fu, J. Zhang, Z. Meng, and I. King, "Magnn: Metapath aggregated graph neural network for heterogeneous graph embedding," in *Proceedings of The Web Conference 2020*, 2020, pp. 2331–2341.
[47] M. He, Z. Wei, and J.-R. Wen, "Convolutional neural networks on graphs with chebyshev approximation, revisited," *Advances in Neural Information Processing Systems*, vol. 35, pp. 7264–7276, 2022.
[48] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv:1710.10903*, 2017.
[49] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv:1810.00826*, 2018.

SUPPLEMENTARY MATERIALS

A. Notations

We dedicate Table XI to index the notations used in this paper. Note that every notation is also defined when it is introduced.

I. THEORETICAL ANALYSIS

A. Bounding #common edges wrt. true subgraph

Let $\mathcal{E}^*$ and $\tilde{\mathcal{E}}$ denote the ordered collection of edges sampled by the idealized learning ORACLE according to true distribution p^* and by SGS−GNN according to learned probability $\tilde{p}$ respectively. For analytical convenience, let us assume that both learning algorithms sample $k = \lfloor q|\mathcal{E}|/100 \rfloor$ edges with replacement independently.

First, we will prove lemma I.1, which show that the probability of an edge chosen by SGS−GNN coincides with that chosen by the ORACLE has a lower bound. Finally, we will prove one of the main results (Theorem I-A), which shows that given $q \in [0, 100]$, we can lower-bound the expected number of common edges between SGS−GNN and the learning ORACLE.

Lemma I.1. *For any arbitrarily chosen $i \in \{1, 2, \ldots, k\}$*

$$\mathbf{Pr}(\mathcal{E}_i^* = \tilde{\mathcal{E}}_i) \geq \sum_{j=1}^{|\mathcal{E}|} \frac{(p_j^* + \tilde{p}_j - \epsilon)^2}{4},$$

where $k = \lfloor q|\mathcal{E}|/100 \rfloor$ and $0 \leq q \leq 100$ is a user-specified parameter and $\epsilon \in [0, 1]$ is the error.

Proof. We prove the above lemma in two parts.

a) Part 1: Universal approximation of probability distribution over edges.: The Universal Approximation Theorem [36], [37] states that a feed-forward neural network with at least one hidden layer and a finite number of neurons can approximate any continuous function $f : \mathbb{R}^n \to \mathbb{R}$ on a compact subset of $\mathbb{R}^n$, given a suitable choice of weights and activation functions.

In our case, $p^* = f$ is the true edge probability distribution for the downstream task, $\tilde{p} = \texttt{EdgePE}_\phi(\mathcal{G})$ is the learned approximate distribution and $\boldsymbol{x}_e$ is a vector of edge features, for instance, $\boldsymbol{x}_e = ((\boldsymbol{h}_u - \boldsymbol{h}_v) \oplus (\boldsymbol{h}_u \odot \boldsymbol{h}_v))$ as used in equation 8. The following universal approximation property holds for the module I component of SGS−GNN,

$$\sup_{e \in \mathcal{E}} \|\tilde{p}(\boldsymbol{x}_e) - p^*(\boldsymbol{x}_e)\|_1 \leq \epsilon. \tag{18}$$

Here, we have two underlying assumptions: (i) the optimal distribution p^* is a function of node features $\boldsymbol{X}$ and (ii) $\boldsymbol{X}$ is a compact subset (bounded and closed) of Euclidean space $\mathbb{R}^n$. The first assumption is made to simplify the problem. The second assumption is quite practical since the node features are typically normalized. Hence, we can show that the embeddings $\boldsymbol{h}_u, \boldsymbol{h}_v$, which are continuous images of $\boldsymbol{X}$, are also compact due to the extreme value theorem. As a result, the edge features $\boldsymbol{x}_e$ which, in a sense, *lifts* the end-point node features into higher-dimensional Euclidean space are also compact. The approximation error ϵ can be made arbitrarily small by increasing the capacity of the MLP, e.g., adding more neurons or layers.

b) Part 2: Common edges wrt. optimal subgraph.: The event $\mathcal{E}_i^* = \tilde{\mathcal{E}}_i$ means that both $\mathcal{E}_i^*$ and $\tilde{\mathcal{E}}_i$ contain the same edge. But there are $|\mathcal{E}|$ such candidates. Hence, the probability of this event is given by,

$$\begin{aligned}
\mathbf{Pr}(\mathcal{E}_i^* = \tilde{\mathcal{E}}_i) &= \sum_{j=1}^{|\mathcal{E}|} \mathbf{Pr}(\mathcal{E}_i^* = \mathcal{E}_j \wedge \tilde{\mathcal{E}}_i = \mathcal{E}_j), \\
&= \sum_{j=1}^{|\mathcal{E}|} \mathbf{Pr}(\mathcal{E}_i^* = \mathcal{E}_j) \cdot \mathbf{Pr}(\tilde{\mathcal{E}}_i = \mathcal{E}_j), \\
&= \sum_{j=1}^{|\mathcal{E}|} p_j^* \cdot \tilde{p}_j, \\
&\geq \sum_{j=1}^{|\mathcal{E}|} \frac{(p_j^* + \tilde{p}_j - |p_j^* - \tilde{p}_j|)^2}{4}, \\
&\geq \sum_{j=1}^{|\mathcal{E}|} \frac{(p_j^* + \tilde{p}_j - \epsilon)^2}{4}.
\end{aligned}$$

The second line follows since the optimal sampler is a different algorithm independent from the sampler used in SGS−GNN. The last line follows because $\|p_j^* - \tilde{p}_j\|_1 \leq \epsilon \implies |p_j^* - \tilde{p}_j| \leq \epsilon$ (from eq. 18). □

We have the following theorem that lower-bounds the number of common edges with respect to the optimal sampler $|\mathcal{E}^* \cap \tilde{\mathcal{E}}|$:

Theorem I.2 (Lower-bound)**.**

$$\mathbb{E}[|\mathcal{E}^* \cap \tilde{\mathcal{E}}|] \geq k \sum_{j=1}^{|\mathcal{E}|} \frac{(p_j^* + \tilde{p}_j - \epsilon)^2}{4}, \tag{19}$$

where $k = \lfloor q|\mathcal{E}|/100 \rfloor$ and $0 \leq q \leq 100$ is a user-specified parameter.

Proof. Since we are drawing k edges independently at random, the theorem follows by applying the linearity of expectation on the following:

$$\begin{aligned}
\mathbb{E}[|\mathcal{E}^* \cap \tilde{\mathcal{E}}|] = \mathbb{E}[\sum_{i=1}^{k} \mathbb{I}(\mathcal{E}_i^* = \tilde{\mathcal{E}}_i)] &= \sum_{i=1}^{k} \mathbf{Pr}(\mathcal{E}_i^* = \tilde{\mathcal{E}}_i) \\
&= k \cdot \mathbf{Pr}(\mathcal{E}_i^* = \tilde{\mathcal{E}}_i) \\
&\geq k \sum_{j=1}^{|\mathcal{E}|} \frac{(p_j^* + \tilde{p}_j - \epsilon)^2}{4}
\end{aligned}$$

□

This theorem shows that the expected number of common edges between the sample subgraph obtained by SGS−GNN $\tilde{\mathcal{G}}$ and the true optimal sample subgraph $\mathcal{G}^*$ is non-trivial.

TABLE XI
NOTATIONS.

$\mathcal{G}$	$\triangleq$	Input graph with a vertex set $\mathcal{V}$, an edge set $\mathcal{E}$, and features $\boldsymbol{X}$
$\boldsymbol{A}$	$\triangleq$	Adjacency matrix of $\mathcal{G}$
$\mathcal{E}$	$\triangleq$	Edges of $\mathcal{G}$
$\mathcal{V}$	$\triangleq$	Nodes of $\mathcal{G}$
$\boldsymbol{X}$	$\triangleq$	Matrix containing node features of $\mathcal{G}$
$\boldsymbol{y}$	$\triangleq$	Vector of node labels of $\mathcal{G}$
C	$\triangleq$	An ordered set containing all possible node labels of $\mathcal{G}$
F	$\triangleq$	Dimension of node features in $\mathcal{G}$
L	$\triangleq$	Number of GNN layers
H	$\triangleq$	Node embedding dimension
$\boldsymbol{H}$	$\triangleq$	Node embedding matrix
$\boldsymbol{h}_u$	$\triangleq$	Embedding of node u
$\boldsymbol{w}$	$\triangleq$	Vector of edge weights in $\mathcal{G}$
q	$\triangleq$	Ratio of # edges in sparse graph and # edges in input graph in %
k	$\triangleq$	# edges in the sparse graph, $k \triangleq \lfloor \frac{q\lvert\mathcal{E}\rvert}{100} \rfloor$
$\tilde{p}$	$\triangleq$	Learned probability distribution by SGS−GNN
$\tilde{\mathcal{E}}$	$\triangleq$	Set of edges sampled from $\mathcal{E}$ by SGS−GNN following $\tilde{p}$
$\tilde{\mathcal{G}}$	$\triangleq$	Sparse subgraph $(\mathcal{V}, \tilde{\mathcal{E}}, \boldsymbol{X})$ constructed by SGS−GNN
$\boldsymbol{A}_{\tilde{\mathcal{G}}}$ or $\tilde{\boldsymbol{A}}$	$\triangleq$	Adjacency matrix of $\tilde{\mathcal{G}}$
$\tilde{\boldsymbol{w}}$	$\triangleq$	Edge weight of sparse graph learned by SGS−GNN
p_{prior}	$\triangleq$	Probability distribution of a fixed prior on $\mathcal{G}$
$\tilde{p}_a$	$\triangleq$	Augmented learned probability distribution
p^*	$\triangleq$	True probability distribution known by the idealized learning ORACLE
$\mathcal{E}^*$	$\triangleq$	Set of edges sampled from $\mathcal{E}$ by the learning ORACLE following distribution p^*
$\mathcal{G}^*$	$\triangleq$	True sparse subgraph $(\mathcal{V}, \mathcal{E}^*, \boldsymbol{X})$ constructed by the learning ORACLE
$\boldsymbol{A}_{\mathcal{G}^*}$ or $\boldsymbol{A}^*$	$\triangleq$	Adjacency matrix of $\mathcal{G}^*$
$\mathcal{L}_{\text{CE}}$	$\triangleq$	Cross entropy loss
$\mathcal{L}_{\text{assor}}$	$\triangleq$	Assortative loss
$\mathcal{L}_{\text{cons}}$	$\triangleq$	Consistency loss
$\mathcal{L}$	$\triangleq$	Total loss

Theorem I.3 (Upper-bound)**.**

$$\mathbb{E}[|\mathcal{E}^* \cap \tilde{\mathcal{E}}|] \leq k(1 - \frac{\|p^* - \tilde{p}\|_1}{2}), \tag{20}$$

where $k = \lfloor q|\mathcal{E}|/100 \rfloor$ and $0 \leq q \leq 100$ is a user-specified parameter.

Proof.

$$\begin{aligned}
\mathbf{Pr}(\mathcal{E}_i^* = \tilde{\mathcal{E}}_i) &= \sum_{j=1}^{|\mathcal{E}|} p_j^* \cdot \tilde{p}_j \\
&\leq \sum_{j=1}^{|\mathcal{E}|} \min(p_j^*, \tilde{p}_j) \\
&= 1 - d_{TV}(p^*, \tilde{p}) \\
&= 1 - \frac{1}{2}\|p^* - \tilde{p}\|_1
\end{aligned}$$

□

Here d_{TV} is the total variation distance. The result used in the last line regarding d_{TV} can be found in [38].

c) The implication of the upper-bound.: When $\tilde{p} \to p^*$, the norm $\|p^* - \tilde{p}\|_1 \to 0$; therefore, the number of common edges could be close to k.

B. Upper-bounding the error in the learned Adjacency matrix

With the bound proven earlier on the #common edges by the sparse subgraph of SGS−GNN with that by a learning ORACLE, in this section, we want to obtain an upper-bound on the error in terms of the norm of the Adjacency matrices. As adjacency matrices are used by GNNs for computing node embeddings, such result is important for obtaining error bound on the embeddings later on.

Let $\boldsymbol{A}_{\tilde{\mathcal{G}}}$ and $\boldsymbol{A}_{\mathcal{G}^*}$ be the corresponding adjacency matrices of the learned sparse graph $\tilde{\mathcal{G}}$ and true optimal sparse graph $\mathcal{G}^*$. The dimension of these matrices is the same as the input adjacency matrix $\boldsymbol{A}_{\mathcal{G}}$ except that $\boldsymbol{A}_{\mathcal{G}}$ is denser. Let us also denote the Frobenius norm of a matrix $\boldsymbol{A}$ as $\|\boldsymbol{A}\|_F$ and the spectral norm of $\boldsymbol{A}$ as $\|\boldsymbol{A}\|_2$. The Frobenius norm of $\boldsymbol{A}$ is defined as $\sqrt{\sum_{ij} \boldsymbol{A}_{ij}^2}$, whereas the spectral norm of $\boldsymbol{A}$ is the largest singular value $\sigma_{max}(\boldsymbol{A})$ of $\boldsymbol{A}$.

Since SGS−GNN do not know the true probability distribution p^*, error is introduced in the learned adjacency matrix

$\boldsymbol{A}_{\tilde{\mathcal{G}}}$ of the downstream sparse subgraph. We are interested in analyzing the expected error introduced in $\boldsymbol{A}_{\tilde{\mathcal{G}}}$ in terms of the spectral norm, to be precise, $\mathbb{E}[\|\boldsymbol{A}_{\tilde{\mathcal{G}}} - \boldsymbol{A}_{\mathcal{G}^*}\|_2]$. To this end, we will exploit the lower bound derived in Theorem 1 and the fact that $\|\boldsymbol{A}\|_2 \leq \|\boldsymbol{A}\|_F$.

Lemma I.4 (Error in Adjacency matrix approximation)**.** *Let $\boldsymbol{A}_{\tilde{\mathcal{G}}}$ and $\boldsymbol{A}_{\mathcal{G}^*}$ be the corresponding adjacency matrices of the learned sparse graph $\tilde{\mathcal{G}}$ and true optimal sparse graph $\mathcal{G}^*$. If the downstream sampler sampled k edges independently at random (with replacement) to construct those matrices following their respective distributions $\tilde{p}$ and p^*, then*

$$\mathbb{E}[\|\boldsymbol{A}_{\tilde{\mathcal{G}}} - \boldsymbol{A}_{\mathcal{G}^*}\|_2] \leq \sqrt{2k(1 - \sum_{j=1}^{|\mathcal{E}|} \frac{(p_j^* + \tilde{p}_j - \epsilon)^2}{4})},$$

where $k = \lfloor q|\mathcal{E}|/100 \rfloor$ and $0 \leq q \leq 100$ is a user-specified parameter.

Proof. Since the entries in adjacency matrices are either 0 or 1, the difference $\boldsymbol{A}_{\tilde{\mathcal{G}}}(i,j) - \boldsymbol{A}_{\mathcal{G}^*}(i,j)$ are in $\{-1, 0, 1\}$ for all i, j. The following holds by definition of Frobenus norm,

$$\|\boldsymbol{A}_{\tilde{G}} - \boldsymbol{A}_{G^*}\|_F^2 = \sum_{ij} (\boldsymbol{A}_{\tilde{\mathcal{G}}}(i,j) - \boldsymbol{A}_{\mathcal{G}^*}(i,j))^2.$$

As a result, only the non-zero entries in $\boldsymbol{A}_{\tilde{\mathcal{G}}} - \boldsymbol{A}_{\mathcal{G}^*}$ contribute to the square of Frobenius norm $\|\boldsymbol{A}_{\tilde{G}} - \boldsymbol{A}_{G^*}\|_F^2$. The expected number of non-zero entries in $\|\boldsymbol{A}_{\tilde{\mathcal{G}}} - \boldsymbol{A}_{\mathcal{G}^*}\|_F^2$ corresponds to the expected cardinality $|(\tilde{\mathcal{E}} \setminus \mathcal{E}^*) \cup (\mathcal{E}^* \setminus \tilde{\mathcal{E}})|$. Thus

$$\begin{aligned}
\mathbb{E}[\|\boldsymbol{A}_{\tilde{\mathcal{G}}} - \boldsymbol{A}_{\mathcal{G}^*}\|_F^2] &= \mathbb{E}[|(\tilde{\mathcal{E}} \setminus \mathcal{E}^*) \cup (\tilde{\mathcal{E}} \setminus \mathcal{E}^*)|] \\
&= \mathbb{E}[|\tilde{\mathcal{E}}| + |\mathcal{E}^*| - 2|\tilde{\mathcal{E}} \cap \mathcal{E}^*|] \\
&= 2k - 2\mathbb{E}[|\tilde{\mathcal{E}} \cap \mathcal{E}^*|] \\
&\leq 2k - 2k \sum_{j=1}^{|\mathcal{E}|} \frac{(p_j^* + \tilde{p}_j - \epsilon)^2}{4} \\
&= 2k(1 - \sum_{j=1}^{|\mathcal{E}|} \frac{(p_j^* + \tilde{p}_j - \epsilon)^2}{4}).
\end{aligned}$$

Applying Jensen's inequality for convex functions, in particular, applying $(\mathbb{E}[\mathrm{X}])^2 \leq \mathbb{E}[\mathrm{X}^2])$ yields,

$$\begin{aligned}
(\mathbb{E}[\|\boldsymbol{A}_{\tilde{\mathcal{G}}} - \boldsymbol{A}_{\mathcal{G}^*}\|_F])^2 &\leq \mathbb{E}[\|\boldsymbol{A}_{\tilde{\mathcal{G}}} - \boldsymbol{A}_{\mathcal{G}^*}\|_F^2] \\
&\leq 2k(1 - \sum_{j=1}^{|\mathcal{E}|} \frac{(p_j^* + \tilde{p}_j - \epsilon)^2}{4}).
\end{aligned}$$

Taking square-root on both sides yields,

$$\mathbb{E}[\|\boldsymbol{A}_{\tilde{\mathcal{G}}} - \boldsymbol{A}_{\mathcal{G}^*}\|_F] \leq \sqrt{2k(1 - \sum_{j=1}^{|\mathcal{E}|} \frac{(p_j^* + \tilde{p}_j - \epsilon)^2}{4})}.$$

We obtain the theorem using the following relation between the Frobenius and spectral norms.

$$\|\boldsymbol{A}_{\tilde{\mathcal{G}}} - \boldsymbol{A}_{\mathcal{G}^*}\|_2 \leq \|\boldsymbol{A}_{\tilde{\mathcal{G}}} - \boldsymbol{A}_{\mathcal{G}^*}\|_F$$

$$\begin{aligned}
\implies \mathbb{E}[\|\boldsymbol{A}_{\tilde{\mathcal{G}}} - \boldsymbol{A}_{\mathcal{G}^*}\|_2] &\leq \mathbb{E}[\|\boldsymbol{A}_{\tilde{\mathcal{G}}} - \boldsymbol{A}_{\mathcal{G}^*}\|_F] \\
&= \sqrt{2k(1 - \sum_{j=1}^{|\mathcal{E}|} \frac{(p_j^* + \tilde{p}_j - \epsilon)^2}{4})}
\end{aligned}$$

□

C. Upper-bounding the error in the predicted node embeddings

We consider vanilla GCN as proof of concept to understand how the changes in the sparse subgraph affect the node embeddings produced by a trained GCN. Our goal is to analyze the respective encodings produced by an L-layer GCN when the input subgraphs are $\mathcal{G}^*$ (corresponding to $\boldsymbol{A}_{\mathcal{G}^*}$) and $\tilde{\mathcal{G}}$ (corresponding to $\boldsymbol{A}_{\tilde{\mathcal{G}}}$) respectively. For simplicity, we will shorten the matrices $\boldsymbol{A}_{\mathcal{G}^*}$ as $\boldsymbol{A}^*$ and $\boldsymbol{A}_{\tilde{\mathcal{G}}}$ as $\tilde{\boldsymbol{A}}$.

A single GCN layer is defined as,

$$\boldsymbol{H}^{(l+1)} = \sigma(\hat{\boldsymbol{A}}\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)}),$$

where $\hat{\boldsymbol{A}} = \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$ is the normalized adjacency matrix, $\boldsymbol{H}^{(l)}$ is the input to the l-th layer with $\boldsymbol{H}^{(0)} = \boldsymbol{X}$, $\boldsymbol{W}^{(l)}$ is the learnable weight matrix for l-th layer and σ is non-linear activation function. Let us suppose an L-layer GCN produces embeddings $\tilde{\boldsymbol{H}}^{(L)}$ and $\boldsymbol{H}^{*(L)}$ when it takes sparse matrices $\tilde{\boldsymbol{A}}$ and $\boldsymbol{A}^*$ as input. We want to upper-bound,

$$\mathbb{E}[\|\tilde{\boldsymbol{H}}^{(L)} - \boldsymbol{H}^{*(L)}\|_2],$$

in other words, the loss in the downstream node encodings is due to using our learned subgraph.

a) Assumptions.: We assume that for all l, $\|\boldsymbol{W}\|_2 \leq \alpha < 1$ where α is a constant no more than 1. This is reasonable since each $\boldsymbol{W}^{(l)}$ is typically controlled during training using regularization techniques, e.g., weight decay. Assuming that the input features in $\boldsymbol{X}$ are bounded, we can also assume that there exists a constant β such that $\forall l$, $\|H\|_2^{(l)} \leq \beta$. We also assume that σ is *Lipschitz continuous* with *Lipschitz constant* L_σ; for instance, activation functions such as `ReLU`, sigmoid, or tanH are Lipschitz continuous. In particular, we assume `ReLU` activation for our theoretical analysis because `ReLU` has *Lipschitz constant* $L_\sigma = 1$, which simplifies our analysis.

Under these assumptions, we have the following theorem,

Theorem I.5 (Error in GCN encodings)**.** *For sufficiently deep L-layer GCN (large L), the error*

$$\mathbb{E}[\lim_{L\to\infty} \|\tilde{\boldsymbol{H}}^{(L)} - \boldsymbol{H}^{*(L)}\|_2] < \frac{\beta}{1-\alpha} \sqrt{2k(1 - \sum_{j=1}^{|\mathcal{E}|} \frac{(p_j^* + \tilde{p}_j - \epsilon)^2}{4})}.$$

Proof.

$$\tilde{\boldsymbol{H}}^{(L)} - \boldsymbol{H}^{*(L)} = \sigma(\hat{\tilde{\boldsymbol{A}}}\tilde{\boldsymbol{H}}^{(L-1)}\boldsymbol{W}^{(L-1)}) - \sigma(\hat{\boldsymbol{A}}^*\boldsymbol{H}^{*(L-1)}\boldsymbol{W}^{(L-1)})$$

Since σ is a Lipschitz continuous function, we have

$$\begin{aligned}
\|\tilde{\boldsymbol{H}}^{(L)} - \boldsymbol{H}^{*(L)}\|_2 &\leq L_\sigma \|\hat{\tilde{\boldsymbol{A}}}\tilde{\boldsymbol{H}}^{(L-1)}\boldsymbol{W}^{(L-1)} - \hat{\boldsymbol{A}}^*\boldsymbol{H}^{*(L-1)}\boldsymbol{W}^{(L-1)}\|_2 \\
&= \|\hat{\tilde{\boldsymbol{A}}}\tilde{\boldsymbol{H}}^{(L-1)}\boldsymbol{W}^{(L-1)} - \hat{\boldsymbol{A}}^*\boldsymbol{H}^{*(L-1)}\boldsymbol{W}^{(L-1)}\|_2 \\
&= \|(\hat{\tilde{\boldsymbol{A}}} - \hat{\boldsymbol{A}}^*)\tilde{\boldsymbol{H}}^{(L-1)}\boldsymbol{W}^{(L-1)} + \hat{\boldsymbol{A}}^*(\tilde{\boldsymbol{H}}^{(L-1)} - \boldsymbol{H}^{*(L-1)})\boldsymbol{W}^{(L-1)}\|_2
\end{aligned}$$

For notational convenience, let us suppose $D^{(L)} = \|\tilde{\boldsymbol{H}}^{(L)} - \boldsymbol{H}^{*(L)}\|_2$. Applying the sub-multiplicative property of the

spectral norm and triangle inequality, we obtain the following recurrence relation

$$\begin{aligned} D^{(L)} &\leq \|(\hat{\tilde{\boldsymbol{A}}} - \hat{\boldsymbol{A}}^*)\|_2 \|\tilde{\boldsymbol{H}}^{(L-1)}\|_2 \|\boldsymbol{W}^{(L-1)}\|_2 + \|\hat{\boldsymbol{A}}^*\|_2 D^{(L-1)} \|\boldsymbol{W}^{(L-1)}\|_2 \\ &\leq \|(\hat{\tilde{\boldsymbol{A}}} - \hat{\boldsymbol{A}}^*)\|_2 \beta\alpha + \|\hat{\boldsymbol{A}}^*\|_2 D^{(L-1)} \alpha \\ &\leq \|(\hat{\tilde{\boldsymbol{A}}} - \hat{\boldsymbol{A}}^*)\|_2 \beta\alpha + D^{(L-1)} \alpha \end{aligned}$$

The last inequality holds because normalized adjacency matrix satisfies $\|\hat{\boldsymbol{A}}^*\|_2 \leq 1$. This is because $\hat{\boldsymbol{A}}^*$ is symmetric, row-stochastic matrix. Thus the singular values of $\hat{\boldsymbol{A}}^*$ is the absolute values of eigenvalues of $\hat{\boldsymbol{A}}^*$ and the largest singular value of $\hat{\boldsymbol{A}}^*$ is the largest eigenvalue of $\hat{\boldsymbol{A}}^*$. But $\hat{\boldsymbol{A}}^*$ being row-stochastic, its largest eigenvalue is at most 1 hence $\|\hat{\boldsymbol{A}}^*\|_2 = \sigma_{max}(\hat{\boldsymbol{A}}^*) \leq 1$.

By unrolling the recursion from earlier inequality:

$$D^{(L)} \leq \|(\hat{\tilde{\boldsymbol{A}}} - \hat{\boldsymbol{A}}^*)\|_2 \beta\alpha \sum_{l=0}^{L-1} \alpha^l + D^{(0)} \alpha^L$$

$D^{(0)} = \|\tilde{\boldsymbol{H}}^{(0)} - \boldsymbol{H}^{*(0)}\|_2 = \|\boldsymbol{X} - \boldsymbol{X}\|_2 = 0$. Since $\alpha < 1$, The geometric series simplifies to:

$$\sum_{l=0}^{L-1} \alpha^l = \frac{1 - \alpha^L}{1 - \alpha}$$

$$\lim_{L\to\infty} \sum_{l=0}^{L-1} \alpha^l = \frac{1}{1 - \alpha}$$

Thus our earlier inequality becomes:

$$\lim_{L\to\infty} D^{(L)} \leq \frac{\beta\alpha}{1 - \alpha} \|(\hat{\tilde{\boldsymbol{A}}} - \hat{\boldsymbol{A}}^*)\|_2 < \frac{\beta}{1 - \alpha} \|(\hat{\tilde{\boldsymbol{A}}} - \hat{\boldsymbol{A}}^*)\|_2$$

Taking expectation on both sides gives us our desired result:

$$\begin{aligned} \mathbb{E}[\lim_{L\to\infty} \|\tilde{\boldsymbol{H}}^{(L)} - \boldsymbol{H}^{*(L)}\|_2] = \mathbb{E}[D^{(L}] &< \frac{\beta}{1 - \alpha} \mathbb{E}[\|(\hat{\tilde{\boldsymbol{A}}} - \hat{\boldsymbol{A}}^*)\|_2] \\ &< \frac{\beta}{1 - \alpha} \mathbb{E}[\|(\hat{\boldsymbol{A}} - \boldsymbol{A}^*)\|_2] \\ &= \frac{\beta}{1 - \alpha} \sqrt{2k(1 - \sum_{j=1}^{|\mathcal{E}|} \frac{(p_j^* + \tilde{p}_j - \epsilon)^2}{4})} \end{aligned}$$

□

II. Analyzing the Effectiveness of SGS−GNN with a Synthetic Graph

In this section, we demonstrate and analyze the effectiveness of SGS−GNN with a synthetically generated heterophilic graph.

a) Synthetic Graph: Moon.: The moon dataset has the following properties: number of nodes $|\mathcal{V}| = 150$, number of edges $|\mathcal{E}| = 870$, average degree $d = 5.8$, node homophily $\mathcal{H}_n = 0.2$, edge homophily $\mathcal{H}_e = 0.32$, training/test split = 30%/70%, and 2D coordinates of the points representing the nodes are the node features $\boldsymbol{X}$. The dataset comprises two half-moons representing two communities with 68% edges connecting them as bridge edges.

b) Explaining the Effectiveness of SGS−GNN on Heterophilic graph.: Fig. 6 juxtaposes the input moon graph (Fig. 6, left) and the sparsified moon graph by SGS−GNN (Fig. 6, right). SGS−GNN removes a significant portion of bridge edges, causing an increase in edge homophily from 0.32 to 1.0. As a result, the accuracy of vanilla GCN increased from 80% on the full graph to 100% on the sparsified graph. Since heterophilous edges significantly hinder the node representation learning, SGS−GNN identifies them during training and learns to put less probability mass on such edges for downstream node classification. Due to this learning dynamics, SGS−GNN is more effective on heterophilic graphs such as the Moon graph.

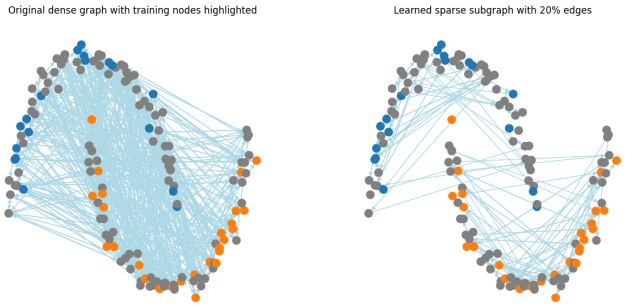

Fig. 6. Toy example with two half-moons demonstrates the effectiveness of SGS−GNN. The original graph has 68% edges with different node labels; in contrast, the learned sparse subgraph from SGS−GNN contains no such bridge edges.

III. Additional Algorithmic Details of SGS−GNN

a) Conditional update of EdgePE.: Backward propagation is often the most computationally intensive part of training, so we employ a conditional mechanism to update EdgePE selectively. We evaluate the learned sparse subgraph (line 9, Alg. 4) against a subgraph from the prior probability distribution p_{prior} (line 11, Alg. 4). If the training F1-score from the learned sparse subgraph is better than the baseline, parameters of EdgePE are updated (line 19, Alg. 4). Otherwise, the update to EdgePE is skipped (line 22, Alg. 4).

The detailed algorithm for SGS−GNN with conditional updates is in Alg. 4.

Algorithm 4 SGS−GNN Training with conditional updates

1: **Input:** $\mathcal{G}(\mathcal{V}, \mathcal{E}, \boldsymbol{X})$, sample percent q, hops, #Parts n
2: Compute $p_{\text{prior}}(u, v) \leftarrow \frac{1/d_u + 1/d_v}{\sum_{i,j \in \mathcal{E}} (1/d_i + 1/d_j)}$
3: $\mathcal{G}_{\text{parts}} = \{\mathcal{G}_1, \mathcal{G}_2, \cdots, \mathcal{G}_n\} \leftarrow \text{METIS}(\mathcal{G}(\mathcal{V}, \mathcal{E}, p), n)$
4: **for** epoch $= 1, 2, \ldots,$ max_epochs **do**
5: **for** $\mathcal{G}_i(\mathcal{V}_i, \mathcal{E}_i, \boldsymbol{X}_i, p^i_{\text{prior}}) \in \mathcal{G}_{\text{parts}}$ **do**
6: $\tilde{p}, \boldsymbol{w} \leftarrow \texttt{EdgePE}_\phi(\mathcal{E}_i, p^i_{\text{prior}}, \boldsymbol{X}_i, \text{hops})$
7: $\tilde{p}_a \leftarrow \lambda \tilde{p} + (1 - \lambda) p^i_{\text{prior}}$
8: $\tilde{\mathcal{E}}, \tilde{\boldsymbol{w}} \leftarrow \text{Sample}(\tilde{p}_a, \boldsymbol{w}, \lfloor \frac{q|\mathcal{E}_i|}{100} \rfloor)$ {**Learned sparse subgraph**}
9: $\hat{\boldsymbol{Y}}, \tilde{\boldsymbol{H}} \leftarrow \text{GNN}_\theta(\tilde{\mathcal{E}}, \tilde{\boldsymbol{w}}, \boldsymbol{X})$
10: $\mathcal{E}_{\text{prior}} \leftarrow \text{Sample}(p_i, \lfloor \frac{q|\mathcal{E}_i|}{100} \rfloor)$ {**Sparse subgraph from prior**}
11: $\hat{\boldsymbol{Y}}_{\text{prior}} \leftarrow \text{GNN}_\theta(\mathcal{E}_{\text{prior}}, \boldsymbol{X})$
12: **if** Evaluate$(\hat{\boldsymbol{Y}}) \geq$ Evaluate$(\hat{\boldsymbol{Y}}_{\text{prior}})$ **then**
13: $\mathcal{L}_{\text{CE}} \leftarrow \text{CrossEntropy}(\boldsymbol{Y}_{\mathcal{V}_L}, \hat{\boldsymbol{Y}}_{\mathcal{V}_L})$
14: $\forall_{(u,v) \in \mathcal{E}_i} u \in \mathcal{V}_L \wedge v \in \mathcal{V}_L : \text{mask[u, v]} \leftarrow \text{True}$
15: $\mathcal{L}_{\text{assor}} \leftarrow \text{CrossEntropy}(\mathcal{E}[\text{mask}], \boldsymbol{w}[\text{mask}])$
16: $\mathcal{L}_{\text{cons}} \leftarrow \text{Sim}(\boldsymbol{w}, \text{Cosine}(\boldsymbol{h}_u, \boldsymbol{h}_v) : \forall_{(u,v) \in \mathcal{E}})$
17: $\mathcal{L} \leftarrow \alpha_1 \cdot \mathcal{L}_{\text{CE}} + \alpha_2 \cdot \mathcal{L}_{\text{assor}} + \alpha_3 \cdot \mathcal{L}_{\text{cons}}$
18: Backward Propagate through $\mathcal{L}$ and optimize $\texttt{EdgePE}_\phi, \text{GNN}_\theta$.
19: **else**
20: $\mathcal{L}_{\text{CE}} \leftarrow \text{CrossEntropy}(\boldsymbol{Y}_{\mathcal{V}_L}, \hat{\boldsymbol{Y}}_{\mathcal{V}_L})$
21: Backward Propagate through $\mathcal{L}_{\text{CE}}$ and optimize GNN_θ.
22: **end if**
23: **end for**
24: **end for**
25: **Return** $\texttt{EdgePE}_\phi, \text{GNN}_\theta$

b) Inference.: During inference, we use the learned probability distribution from EdgePE. We keep track of the best temperature T that gave the best validation accuracy and use that to sample an ensemble of sparse subgraphs. Then, we mean-aggregate their representations to produce the final prediction on a test node.

The reason we consider ensemble of subgraphs is because there are variability in the edges of the sample subgraphs even if they are all sampled from the same distribution. Thus mean-aggregation of node embeddings is an effective way to improve the robustness of the learned node embeddings.

The inference pseudocode is provided in Algorithm 5.

IV. Dataset Description

Table XII shows the details of the characteristics of the graph datasets, including the splits used throughout the experimentation.

Along with synthetic dataset, for heterophily, we used, *Cornell, Texas*, *Wisconsin* from the *WebKB* [16]; *Chameleon*, *Squirrel* [39]; *Actor* [16]; *Wiki, ArXiv-year, Snap-Patents, Penn94, Pokec, Genius, reed98, amherst41, cornell5*, and *Yelp* [40]. We also experiment on some recent benchmark

TABLE XII

ADDITIONAL DETAILS OF THE DATASET ARE PROVIDED. $\mathcal{H}_{\text{adj}}$ REFERS TO ADJUSTED HOMOPHILY. d CORRESPONDS TO THE AVERAGE DEGREE, C NUMBER OF CLASSES, AND F IS THE FEATURE DIMENSION. *Tr.* IS THE TRAINING LABEL RATE. *SL.* - IF A SELF-LOOP EXISTS. *Iso.* - IF ANY ISOLATED VERTICES.

DATASET	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	d	$\mathcal{H}_{\text{adj}}$	C	F	TR.	SL.	ISO.	CONTEXT
CORNELL	183	557	3.04	-0.42	5	1703	0.48	Y	N	WEB PAGES
TEXAS	183	574	3.14	-0.26	5	1703	0.48	Y	N	WEB PAGES
WISCONSIN	251	916	3.65	-0.20	5	1703	0.48	Y	N	WEB PAGES
REED98	962	37,624	39.11	-0.10	3	1001	0.6	N	N	SOCIAL NETWORK
AMHERST41	2,235	181,908	81.39	-0.07	3	1193	0.6	N	N	SOCIAL NETWORK
PENN94	41,554	2,724,458	65.56	-0.06	2	4814	0.47	N	N	SOCIAL NETWORK
ROMAN-EMPIRE	22,662	65,854	2.91	-0.05	18	300	0.5	N	N	WIKIPEDIA
CORNELL5	18,660	1,581,554	84.76	-0.04	3	4735	0.6	N	N	WEB PAGES
SQUIRREL	5,201	396,846	76.30	-0.01	5	2345	0.48	Y	N	WIKIPEDIA
JOHNSHOPKINS55	5,180	373,172	72.04	0.00	3	2406	0.6	N	N	WEB PAGES
ACTOR	7,600	53,411	7.03	0.01	5	932	0.48	Y	N	ACTORS IN MOVIES
MINESWEEPER	10,000	78,804	7.88	0.01	2	7	0.5	N	N	SYNTHETIC
QUESTIONS	48,921	307,080	6.28	0.02	2	301	0.5	N	N	YANDEX Q
CHAMELEON	2,277	62,792	27.58	0.03	5	2581	0.48	Y	N	WIKI PAGES
TOLOKERS	11,758	1,038,000	88.28	0.09	2	10	0.5	N	N	TOLOKA PLATFORM
AMAZON-RATINGS	24,492	186,100	7.60	0.14	5	556	0.5	N	N	CO-PURCHASE NETWORK
GENIUS	421,961	1,845,736	4.37	0.17	2	12	0.6	N	Y	SOCIAL NETWORK
POKEC	1,632,803	44,603,928	27.32	0.42	3	65	0.6	N	N	SOCIAL NETWORK
ARXIV-YEAR	169,343	2,315,598	13.67	0.26	5	128	0.6	N	N	CITATION
SNAP-PATENTS	2,923,922	27,945,092	9.56	0.21	5	269	0.6	Y	Y	CITATION
OGBN-PROTEINS	132,534	79,122,504	597.00	0.05	94	8	0.2	N	N	PROTEIN NETWORK
CORA	19,793	126,842	6.41	0.56	70	8710	0.2	N	N	CITATION NETWORK
DBLP	17,716	105,734	5.97	0.68	4	1639	0.2	N	N	CITATION NETWORK
COMPUTERS	13,752	491,722	35.76	0.68	10	767	0.6	N	Y	CO-PURCHASE NETWORK
PUBMED	19,717	88,648	4.50	0.69	3	500	0.2	N	N	SOCIAL NETWORK
CORA_ML	2,995	16,316	5.45	0.75	7	2879	0.2	N	N	CITATION NETWORK
SMALLCORA	2,708	10,556	3.90	0.77	7	1433	0.05	N	N	CITATION NETWORK
CS	18,333	163,788	8.93	0.78	15	6805	0.2	N	N	CO-AUTHOR NETWORK
PHOTO	7,650	238,162	31.13	0.79	8	745	0.2	N	Y	CO-PURCHASE NETWORK
PHYSICS	34,493	495,924	14.38	0.87	5	8415	0.2	N	N	CO-AUTHOR NETWORK
CITESEER	4,230	10,674	2.52	0.94	6	602	0.2	N	N	CITATION NETWORK
WIKI	11,701	431,726	36.90	0.58	10	300	0.99	Y	Y	WIKIPEDIA
REDDIT	232,965	114,615,892	491.99	0.74	41	602	0.66	N	N	SOCIAL NETWORK

Algorithm 5 `SGS-GNN` Inference

Input: Graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, \boldsymbol{X})$, sample % q, Ensemble size R.
$\boldsymbol{w}, \tilde{p} = \texttt{EdgePE}_{\phi}(\mathcal{E}, \boldsymbol{X}, T_{\text{best}})$ {T **with best validation accuracy**.}
$S_y \leftarrow \emptyset$ {Predictions}
for i in R **do**
  $\tilde{\mathcal{E}}, \tilde{\boldsymbol{w}} \leftarrow \text{Sample}(\tilde{p}, \lfloor \frac{q|\mathcal{E}|}{100} \rfloor)$
  $\tilde{\boldsymbol{Y}}_i \leftarrow \text{GNN}_{\theta}(\tilde{\mathcal{E}}, \tilde{\boldsymbol{w}}, \boldsymbol{X})$
  $S_y \leftarrow S_y \cup \tilde{\boldsymbol{Y}}_i$
end for
Predict, $\tilde{\boldsymbol{Y}} \leftarrow \text{Mean}(S_y)$
Return $\tilde{\boldsymbol{Y}}$

datasets, *Roman-empire, Amazon-ratings, Minesweeper, Tolokers*, and *Questions* from [41].

For homophily, we used *Cora* [42]; *Citeseer* [43]; *pubmed* [44]; *Coauthor-cs*, *Coauthor-physics* [45]; *Amazon-computers*, *Amazon-photo* [45]; *Reddit* [21]; and, *DBLP* [46].

V. ABLATION STUDIES

This section investigates how different components of `SGS-GNN` behave and contribute to overall performance. We organize this section as follows,

1) Section V-A investigates $\mathcal{L}_{\text{assor}}, \mathcal{L}_{cons}$, `EdgePE`, `GNN`, and Conditional Updates mechanism. We also compare its runtime against standard `SGS-GNN` training vs `SGS-GNN` with conditional updates. We also show `SGS-GNN` can be used with other GNNs in Sec V-A1.
2) Section V-B explores parameter settings with/without prior, different normalization and sampling methods, and inference with/without an ensemble of subgraphs.
3) Section V-C shows ideal settings for regularizer coefficients $\alpha_1, \alpha_2, \alpha_3$. We also show the impact of λ for augmenting the learned probability distribution p using p_{prior}.

A. Parameter's sensitivity

$\mathcal{L}_{\text{assor}}, \mathcal{L}_{cons}$, `EdgePE`, `GNN`, and Conditional Updates

Table XIII illustrates the performance of `SGS-GNN` with various combinations of regularizers, embedding layers in `EdgePE`, and convolutional layers in `GNN`.

1) $\mathcal{L}_{\text{assor}}$: Case 1, 2 shows improvement in results when $\mathcal{L}_{\text{assor}}$ is used.
2) $\mathcal{L}_{\text{cons}}$: From cases 6, 7, 8 shows L_{cons} improves results when `GCN` module is used in the GNN.
3) `EdgePE`: In general, we found that the `GCN` layers for `EdgePE` encodings perform best (cases 7, 15).
4) `GNN`: Both `GCN` and `GAT` modules yielded overall the best results (case 7, 15).
5) Conditional updates: Case 3 shows that conditional updates can benefit some graphs.

We also investigated the runtime and quality of `SGS-GNN` with and without conditional updates for large-scale graphs. We found both have similar runtimes as the condition check expense gets compensated by fewer updates of `EdgePE`.

TABLE XIII
COMBINATION OF EDGEPE, GNN, CONDITIONAL UPDATE AND L_{cons}.

CASE	L_{assor}	L_{cons}	EDGEPE	GNN	COND.	SMALLCORA	CORAFULL	JOHNSHOPKIN
1	N	N	MLP	GCN	N	73.80 ± 0.67	61.78 ± 0.20	66.12 ± 1.38
2	Y	N	MLP	GCN	N	74.88 ± 0.15	63.99 ± 0.24	66.18 ± 1.05
3	Y	N	MLP	GCN	Y	75.82 ± 0.46	64.07 ± 0.31	66.87 ± 0.93
4	Y	Y	MLP	GCN	Y	76.58 ± 0.47	65.33 ± 0.28	69.25 ± 0.76
5	Y	Y	MLP	GCN	N	73.90 ± 0.81	64.74 ± 0.21	69.21 ± 0.60
6	Y	N	GCN	GCN	Y	75.80 ± 0.77	65.66 ± 0.14	71.06 ± 0.32
7	Y	Y	GCN	GCN	Y	77.50 ± 0.62	**66.56 ± 0.22**	70.79 ± 0.18
8	Y	Y	GCN	GCN	N	75.88 ± 0.71	65.87 ± 0.17	69.83 ± 0.67
9	Y	N	GSAGE	GCN	Y	75.82 ± 0.44	63.70 ± 0.09	67.53 ± 0.80
10	Y	Y	GSAGE	GCN	Y	77.48 ± 0.61	65.12 ± 0.11	68.63 ± 0.66
11	Y	Y	GSAGE	GCN	N	75.44 ± 1.40	64.70 ± 0.21	68.31 ± 0.50
12	Y	N	MLP	GAT	Y	77.72 ± 1.63	66.40 ± 0.08	67.92 ± 0.73
13	Y	Y	MLP	GAT	Y	75.78 ± 3.22	66.46 ± 0.16	68.17 ± 0.33
14	Y	Y	MLP	GAT	N	74.17 ± 2.47	65.89 ± 0.31	67.82 ± 0.40
15	Y	N	GCN	GAT	Y	**78.18 ± 0.74**	66.33 ± 0.20	**71.97 ± 0.59**
16	Y	Y	GCN	GAT	Y	76.94 ± 2.76	66.39 ± 0.18	71.00 ± 0.96
17	Y	Y	GCN	GAT	N	75.57 ± 1.55	65.70 ± 0.38	71.62 ± 0.86
18	Y	N	GSAGE	GAT	Y	77.98 ± 0.79	66.38 ± 0.23	69.29 ± 1.56
19	Y	Y	GSAGE	GAT	Y	75.74 ± 2.02	66.41 ± 0.25	68.82 ± 0.24
20	Y	Y	GSAGE	GAT	N	74.33 ± 2.01	65.22 ± 0.21	66.31 ± 0.52

1) SGS−GNN with other GNN modules: The sampled sparse subgraphs from EdgePE can be fed into any downstream GNNs and demonstrate a couple of variants of SGS−GNN. Chebnet from Chebyshev [47], Graph Attention Network (GAT) [48], Graph Isomorphic Network (GIN) [49], Graph Convolutional Network (GCN) [15] are some of the GNNs used for demonstration.

Fig. 7 shows the performance of these GNNs on homophilic and heterophilic datasets. SGS−GCN and SGS−GAT are two best performing models.

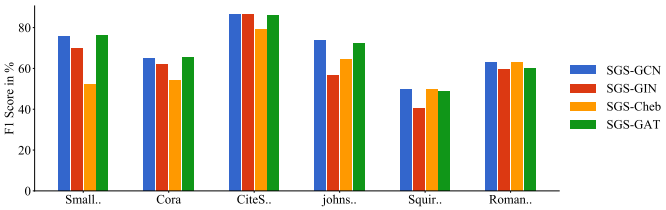

Fig. 7. Performance of SGS−GNN with different GNN modules using 20% edges.

B. Impact of components

Impact of p_{prior}, Normalization & Sampling schemes, and Ensembling on SGS−GNN. Table XIV highlights the impact of the following components:

1) Prior p_{prior}: Cases 2-3 show that augmenting the learned probability distribution $\tilde{p}$ with prior p_{prior} can benefit some datasets. We have also conducted an in-depth comparison between the distributions $\tilde{p}$ and augmented distribution $\tilde{p}_a$. Figure 8 shows that there $\tilde{p}_a$ is left skewed whereas $\tilde{p}$ is not. Since rare edges still get some negligible mass, it is possible for $\tilde{\mathcal{G}}$ constructed from $\tilde{p}_a$ to retain some bridge edges from these tails, if there are any.
2) Normalization and Sampling: We considered three normalization and sampling techniques. i) sum-normalization with multinomial sampling, ii) softmax-normalization with temperature with multinomial sampling, and iii) Gumbel softmax normalization with Topk selection. Cases 3-5 show that each of these techniques can improve results in certain datasets, and thus, it is difficult to nominate a single one as best. However, in our experiments, we opted for multinomial sampling with softmax temperature annealing for training to encourage exploration in early iterations.
3) Ensemble subgraphs during inference: Case 2 demonstrates that using multiple subgraphs for ensemble prediction yields better results than a single subgraph (Case 1).

C. Gridsearch for values of parameters.

Here we find the values for regularizer coefficient α_3 and Parameter λ. SGS−GNN computes the total loss at each epoch as

$$\mathcal{L} = \alpha_1 \mathcal{L}_{CE} + \alpha_2 \mathcal{L}_{\text{assor}} + \alpha_3 \mathcal{L}_{\text{cons}},$$

where $0 \leq \alpha_1, \alpha_2, \alpha_3 \leq 1$ are regularizer coefficients corresponding to the cross-entropy loss $\mathcal{L}_{CE}$, assortativity loss L_{assor} and consistency loss $\mathcal{L}_{\text{cons}}$ respectively.

Also recall that, when we use a prior probability distribution, the learned distribution values of $\tilde{p}$ are weighted through λ in $\tilde{p} = \lambda\tilde{p} + (1-\lambda)p_{\text{prior}}$

To avoid numerous combinations of values of three coefficients + the parameter λ, we have fixed $\alpha_1 = 1$, and $\alpha_2 = 1$. In the following, we investigate the performance of SGS−GNN with different values for α_3 and λ.

Fig. 9 shows a grid search for different combinations of λ and α_3. As per our observation, the recommended values are $\lambda \in [0.3, 0.7], \alpha_3 = 0.5$.

TABLE XIV
ABLATION STUDIES DIFFERENT COMPONENTS OF SGS-GNN.

CASE	PRIOR	NORM.	SAMPL.	ENSEM.	SMALLCORA	CORA_ML	CITESEER	SQUIRREL	JOHNSHOPKINS55	ROMAN-EMPIRE
1	N	SUM	MULT	N	69.30 ± 1.20	81.05 ± 0.74	82.84 ± 0.47	48.90 ± 1.06	63.86 ± 0.58	63.27 ± 0.31
2	N	SUM	MULT	Y	72.84 ± 0.91	82.92 ± 0.73	**87.42 ± 0.42**	46.30 ± 1.18	65.14 ± 1.14	**64.31 ± 0.13**
3	Y	SUM	MULT	Y	75.54 ± 0.41	**83.87 ± 0.69**	86.31 ± 0.26	47.97 ± 0.60	72.68 ± 0.51	62.88 ± 0.19
4	Y	SOFTMAX	MULT	Y	75.44 ± 0.51	83.81 ± 0.72	86.31 ± 0.26	47.90 ± 0.42	**72.97 ± 0.20**	62.98 ± 0.16
5	Y	GUMBEL	TOPK	Y	**76.24 ± 0.43**	83.36 ± 0.34	86.44 ± 0.16	**51.49 ± 0.72**	71.83 ± 1.00	63.00 ± 0.11

PRIOR: USE OF PRIOR, **SUM:** SUM-NORMALIZATION, **SOFTMAX:** *Softmax* WITH TEMPERATURE ANNEALING
MULT: *Multinonmial* SAMPLING, **GUMBEL:** *Gumbel-Softmax* WITH *TopK*

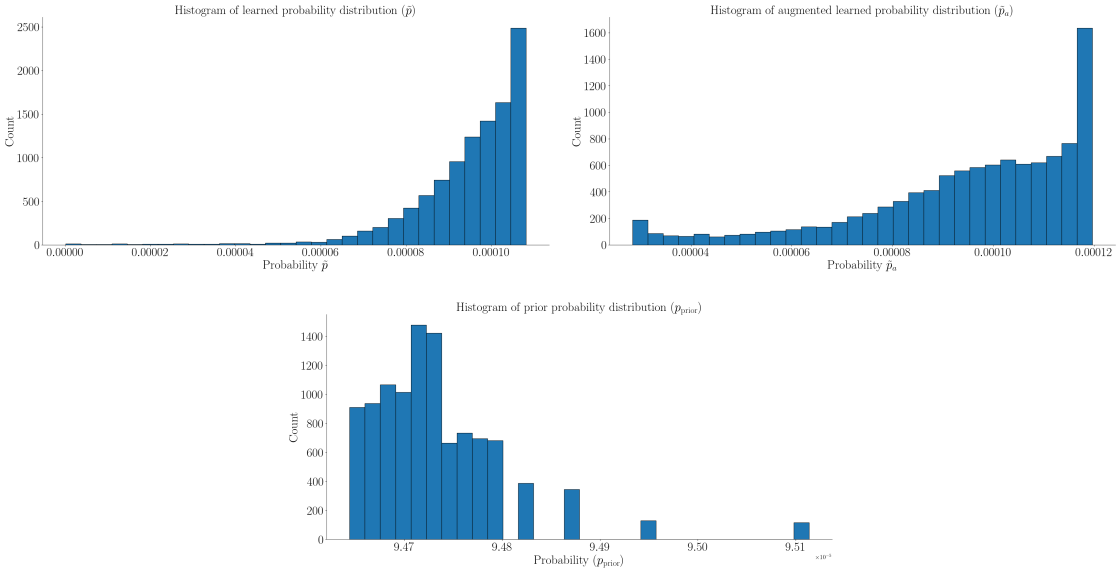

Fig. 8. The learned probability distribution $\tilde{p}$ (top-left), augmented distribution $\tilde{p}_a$(top-right) and fixed prior p_{prior} (bottom). Augmentation puts negligible mass on some rare yet critical edges in the left tail of $\tilde{p}_a$.

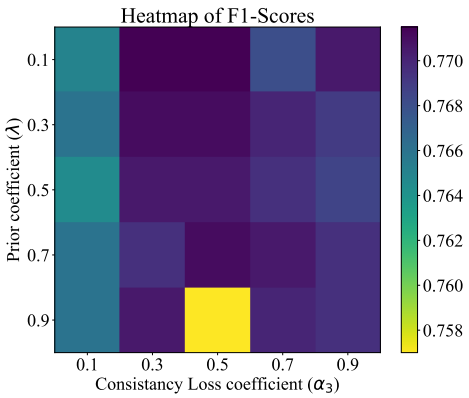

Fig. 9. Grid search for the parameter λ for prior, and consistency loss, α_3 (Cora dataset).