GRAPES: LEARNING TO SAMPLE GRAPHS FOR SCALABLE GRAPH NEURAL NETWORKS

Taraneh Younesian[1], **Thiviyan Thanapalasingam**[2], **Emile van Krieken**[1,3]
Daniel Daza[1,2,4] **& Peter Bloem**[1]
[1]Vrije Universiteit Amsterdam
[2]University of Amsterdam
[3]University of Edinburgh
[4]Discovery Lab, Elsevier, The Netherlands
`{t.younesian,d.dazacruz,p.bloem}@vu.nl`
`thiviyan.t@gmail.com`
`Emile.van.Krieken@ed.ac.uk`

ABSTRACT

Graph neural networks (GNNs) learn the representation of nodes in a graph by aggregating the neighborhood information in various ways. As these networks grow in depth, their receptive field grows exponentially due to the increase in neighborhood sizes, resulting in high memory costs. Graph sampling solves memory issues in GNNs by sampling a small ratio of the nodes in the graph. This way, GNNs can scale to much larger graphs. Most sampling methods focus on fixed sampling heuristics which may not generalize to different structures or tasks. We introduce GRAPES, an *adaptive* graph sampling method that learns to identify sets of influential nodes for training a GNN classifier. GRAPES uses a GFlowNet to learn node sampling probabilities given the classification objectives. We evaluate GRAPES across several small- and large-scale graph benchmarks and demonstrate its effectiveness in accuracy and scalability. In contrast to existing sampling methods, GRAPES maintains high accuracy even with small sample sizes and, therefore, can scale to very large graphs. Our code is publicly available at `https://github.com/dfdazac/grapes`.

1 INTRODUCTION

We represent data with graph structures in many applications. Some examples are recommender systems, social networks, and the chemical and medical domains (Nettleton, 2013; Wu et al., 2022; Li et al., 2022). In these domains, graph neural networks (GNNs) are a powerful tool for representation learning on graphs (Kipf & Welling, 2016; Velickovic et al., 2017; Yun et al., 2019).

Unlike traditional machine learning on i.i.d. data, scalability is a big challenge in machine learning on graphs. One of the reasons is the connectivity of the nodes and their dependence on their neighbors, which makes dividing the data into mini-batches a non-trivial task. Secondly, in each layer of the GNN, the neighbors of the nodes in the previous layer are added to the network. Therefore, the neural network's receptive field increases with the network's depth, resulting in an exponential growth in the number of nodes it needs to process.

Graph sampling tackles the scalability problem in GNNs by considering only a sampled subset of the nodes in the graph. Currently most popular sampling methods are non-adaptive, sampling nodes independently of their features or the training task. Here, we argue for the benefits of *adaptive sampling*, where we let the sampling method adapt to the task and directly sample the nodes that lead to a highly accurate GNN. In particular, we develop a sampler that learns to sample effectively by maximizing GNN accuracy given a fixed sampling budget, i.e., the number of nodes it samples. Most current graph sampling methods instead focus on accurately approximating the behavior of the full-batch GNN (Chen et al., 2018b; Zou et al., 2019; Zeng et al., 2019; Huang et al., 2018), opting for an indirect approach to achieve high accuracy.

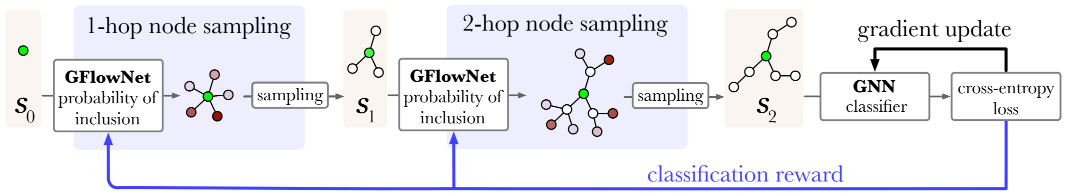

Figure 1: A high-level schematic diagram of GRAPES for a mini-batch of a two-layer classification GNN. GRAPES processes a target node (in green) by retrieving 1-hop neighbors and computing node inclusion probabilities (shown by node color shade) via a GFlowNet. Given these probabilities, GRAPES samples k nodes. This process is repeated over nodes in the 2-hop neighborhood, with each step determining a state s_i. The sampled subgraph is passed to a GNN for target node classification. Classification loss is used to update the GNN classifier and as a reward for the GFlowNet.

To this end, we develop **G**FlowNet **Grap**h N**e**ighbor **S**ampling (GRAPES), an adaptive graph sampling method that uses GFlowNets (Bengio et al., 2021a). Figure 1 shows the general framework of GRAPES. The GFlowNet sampler in GRAPES gets feedback from the GNN to learn scores over the nodes to determine which should be more likely to be included. The GFlowNet framework allows us to consider the sampling process and the GNN computation together and train them in concert. This allows training a GFlowNet sampler that explicitly depends on factors like the features of the nodes, the structure of the graph, sample size, and other contextual features.

We evaluate GRAPES on several node classification tasks and find that

1. GRAPES outperforms state-of-the-art sampling-based methods on most of the datasets.
2. GRAPES is competitive with a state-of-the-art non-sampling GNN scaling method, while using up to an order of magnitude less GPU memory.
3. GRAPES maintains high accuracy at low sample sizes, while performance often drops sharply for the baselines as the sample size is decreased. These results highlight GRAPES' ability to correctly identify influential nodes in the graph and allow scaling to very large graphs while keeping a high performance for the GNN.
4. GRAPES learns to distinguish between nodes and learns different preferences over nodes based on their influence on the performance of the GNN.

2 Related Work

Here, we summarize previous work on improving the scalability of GNN by sampling and give an overview of the related studies using GFlowNets.

2.1 Scalable Graph Learning

Fixed Policy: Most graph sampling methods fall into this category. Here, the sampling policy is independent of the training of the GNN and is based on a fixed heuristic that does not involve any training. Given a set of target nodes, i.e., the nodes that are to be classified, node-wise sampling methods sample a give amount of nodes for each target node. GraphSage (Hamilton et al., 2017) is a node-wise sampling method that randomly samples nodes. However node-wise sampling results in redundantly sampling one node multiple times because it can be the neighbor of several nodes (Zou et al., 2019). A more efficient approach is layer-wise sampling, for example, FastGCN (Chen et al., 2018b) and LADIES (Zou et al., 2019). They aim to minimize variance by sampling nodes in each layer with probabilities proportional to their degree. Moreover, some techniques focus on sampling subgraphs in each mini-batch, like GraphSAINT (Zeng et al., 2019) and ClusterGCN (Chiang et al., 2019). While these techniques are successful in scaling GNNs to larger graphs, they do not adapt to the GNN's performance on the task as they fix the sampling policy.

Learnable Policy: A few methods learn the probability of including a node based on feedback from the GNN. AS-GCN (Huang et al., 2018) is a method that learns a linear function that estimates the node probabilities in a layer-wise manner. Similarly, PASS (Yoon et al., 2021) learns a compositional policy of random and importance-based sampling, including a learnable transformation matrix to learn the similarities between the target node and the candidate neighbors. Finally, GNN-BS (Liu et al., 2020) formulate the node-wise sampling problem as a bandit problem and update the sampling policy according to a reward function that reduces the sampling variance. All these methods focus on variance reduction and fail to consider the classification loss, unlike GRAPES. We argue that adaptivity to the classification loss, allows for sampling the influential nodes depending on the task, and results in better performance.

Historical Embedding: Another group of papers tackling the scalability problem of GNNs use historical embeddings of the nodes when updating the target nodes' embeddings (Chen et al., 2018a; Fey et al., 2021; Yu et al., 2022; Shi et al., 2023). VR-GCN (Chen et al., 2018a) uses historical activations with the aim of variance reduction. GAS (Fey et al., 2021) approximates the embeddings of the 1-hop neighbors using the historical embeddings of those nodes learned in the previous training iterations. These methods reduce the GPU memory usage by training in mini-batches and learning from the 1-hops neighbors with the historical embeddings saved in CPU memory. Unlike GRAPES, they process the whole graph.

2.2 Generative Flow Networks

Generative Flow Networks (GFlowNets) (Bengio et al., 2021a;b) are generative models that sample from a very large structured space with probability proportional to a given reward function. GFlowNets have been used in several applications like molecule design and material science (Bengio et al., 2021a; Gao et al., 2022; Jain et al., 2022), Bayesian structure learning (Deleu et al., 2022), scientific discovery (Jain et al., 2023), and GNN explainability (Li et al., 2023). Similar to our work, the latter utilizes a GFlowNet to sample a subgraph. However, this method *explains* a trained GNN and is not used to scale GNN training to large graphs.

3 Background

3.1 GNN Training and Sampling

Let $\mathcal{G}_C = (\mathcal{V}, \mathcal{E})$ be an undirected graph with a set of N nodes $\mathcal{V}$ and edges $\mathcal{E}$. The adjacency matrix $A \in \{0,1\}^{N \times N}$ indicates a connection between a pair of nodes. $\hat{A} = \tilde{D}^{-1}\tilde{A}$ is the row-normalized adjacency matrix with self-loops, where $\tilde{A} = A + I$ and where $\tilde{D}$ is the degree matrix of $\tilde{A}$. While our method is independent of the choice of GNN architecture, we limit our discussion to the GCN architecture for simplicity (Kipf & Welling, 2016). $X \in \mathbb{R}^{N \times f}$ indicates the given features on the nodes[1] and $Y \in \mathbb{Z}^{N^t}$ is the labels for target nodes $\mathcal{V}^t \subset \mathcal{V}$, the nodes with the labels. The task of node classification aims to learn the labels of these target nodes.

The output of the l-th layer of the GCN is $H^l = \sigma(\tilde{A}H^{l-1}W^l)$, where $W^l \in \mathbb{R}^{d^l \times d^{l+1}}$ is the weight matrix of layer l and σ is a non-linear activation function. d^l is the hidden dimension of layer l, where d^0 is the input feature size and d^L is the number of classes, and L is the number of layers. For a node $v_i \in \mathcal{V}$, this corresponds to the update

$$h^l_{v_i} = \sigma\left(\sum_{v_j \in \mathcal{N}(v_i)} \hat{A}_{v_i,v_j} h^{l-1}_{v_j} W^l\right), \tag{1}$$

where $\mathcal{N}(v_i)$ is the set of v_i's neighbors, and $\hat{A}_{v_i,v_j}$ refers to element (v_i, v_j) of $\hat{A}$.

As the number of layers increases, the computation of the embedding of node v_i involves neighbors from further hops. As a result, the neighborhood size grows exponentially with the number of layers. We study how to sample the graph to overcome this exponential growth. We focus on layer-wise sampling, an effective graph sampling approach (Chen et al., 2018b; Zou et al., 2019; Huang et al., 2018). First, the target nodes are divided into mini-batches of size b. Then, in each layer, k nodes get sampled among the neighbors of the nodes in the previous layer. Therefore, the approximation of the

[1]If no features are available, the feature vectors are set to one-hot vectors as node identifiers.

l-th update of node v_i is:

$$\tilde{h}^l_{v_i} = \sigma\left(\sum_{v_j \in K^l} \hat{A}'^l_{v_i,v_j} \tilde{h}^{l-1}_{v_j} W^l\right), K^l \sim q(K^l | \mathcal{N}(K^{l-1})) \tag{2}$$

where 1) $K^l \subseteq \mathcal{N}(K^{l-1})$ is the set of sampled nodes in layer l, and $\mathcal{N}(K^{l-1})$ indicates the set of neighbors of the nodes in K^{l-1}, 2) K^0 is the set of mini-batch target nodes, 3) $\hat{A}'^l(v_i, v_j)$ is the row normalized value of the sampled adjacency matrix A'^l in layer l [2], where $A'^l(v_i, v_j)$ is 1 if v_j is sampled, i.e., $v_j \in K^l$ and 0 otherwise, and 4) $q(K^l | \mathcal{N}(K^{l-1}))$ is the probability of sampling the set of nodes K^l given the nodes that were sampled in the previous layer and their neighbors. Our goal is to learn the distribution q that maximizes the classification accuracy given a fixed sampling budget of k nodes per layer.

3.2 GFlowNet and Trajectory Balance Loss

We next give a brief overview of GFlowNets (Bengio et al., 2021a;b) and the trajectory balance loss (Malkin et al., 2022a). Let $\mathcal{G}_F = (\mathcal{S}, \mathcal{A}, \mathcal{S}_0, \mathcal{S}_f, R)$ denote a GFlowNet learning problem. Here, $\mathcal{S}$ is a finite set of states that forms a directed graph with $\mathcal{A}$, a set of directed edges representing actions or transitions between states. $\mathcal{S}_0 \subset \mathcal{S}$ is the set of initial states, $\mathcal{S}_f \subset \mathcal{S}$ is the set of terminating states,[3] and $R : \mathcal{S}_f \to \mathbb{R}_+$ is the reward function defined on terminating states. At time t, a particular $a_t \in \mathcal{A}$ indicates the action taken to transition from state s_t to s_{t+1}. A trajectory τ is a path through the graph from an initial state s_0 to a terminating state $s_n \in \mathcal{S}_f$: $\tau = (s_0 \to ... \to s_n)$. A GFlowNet is a neural network that learns to transition from an initial state s_0 to a terminating state where the reward $R(s_n)$ is given. The goal of our GFlowNet is to ensure that the forward transition probabilities $P_F(s_{t+1}|s_t)$ are in proportion to the reward (Bengio et al., 2021a). The *Trajectory Balance (TB)* loss (Malkin et al., 2022a) is developed with this goal. For a trajectory $\tau = (s_0 \to ... \to s_n)$, the TB loss is:

$$\mathcal{L}_{TB}(\tau) = \left(\log \frac{Z(s_0)\prod_{t=1}^n P_F(s_t|s_{t-1})}{R(s_n)\prod_{t=1}^n P_B(s_{t-1}|s_t)}\right)^2, \tag{3}$$

where $Z : \mathcal{S}_0 \to \mathbb{R}_+$ computes the total flow of the network from starting state s_0 and P_F and P_B are the forward and the backward transition probabilities between the states, where both can be parameterised by a neural network (Malkin et al., 2022a).

4 GFlowNet Graph Neighbor Sampling (GRAPES)

In this section, we introduce GFlowNet Graph Neighbor Sampling (GRAPES). GRAPES is a layer-wise and layer-dependent sampling method. In each layer l, it samples $k \ll |\mathcal{N}_{K^{l-1}}| = n$ neighbors of the nodes that were sampled in the previous layer. In the remainder of this section, we describe our GFlowNet design, training and reward function (Section 4.1), adjustments to the reward function (Section 4.2, and our scalable sampling procedure (Section 4.3.

4.1 GFlowNet Design: States, Actions, and Reward

Next, we explain our choice of $\mathcal{G}_F$, that is, the states, actions, terminating states, and reward function, and the coupling of our GFlowNet with the GCN_C. A state $s \in \mathcal{S}$ represent a sequence of adjacency matrices $s = (A_0, ..., A_l)$ sampled so far. An action from s represents choosing k nodes without replacement among the neighbors of the nodes in A_l. This forms the adjacency matrix A_{l+1} of the next layer. Therefore, in an L layer GCN, we construct a sequence of L adjacency matrices to reach a terminating state.

We define the optimal sampling policy as having the lowest classification loss in expectation. Therefore, a set of k nodes with a lower classification loss than another set must have a higher probability.

[2]Unlike the full-batch GCN, when sampling, the adjacency matrix of each layer is different from the other layers because the nodes involved in each layer are different.

[3]Technically, GFlowNets have unique source and terminal (or 'sink') states s_s and s_f. The source state has an edge to all initial states, and all terminating states have an edge to the terminal state.

Our goal is to design $\mathcal{G}_F$ so that it learns the forward and backward transition probabilities proportional to a given reward. We define the reward function as below:

$$R(s_L) = R(A_0, \ldots, A_L) := \exp(-\alpha \cdot \mathcal{L}_{GCN_C}(A_0, \ldots, A_L)), \tag{4}$$

where $\mathcal{L}_{GCN_C}$ is the classification loss and α is a *scaling parameter*, which we explain in Section 4.2. A_0 is the adjacency matrix of the graph given as input at the starting state, and A_1 to A_L are the adjacency matrices sampled in each transition. We explain the sampling procedure in section 4.3.

Forward Probability. The GFLowNet of GRAPES is a neural network that learns the forward probabilities $P_F(s_l|s_{l+1})$, the probability of sampling an adjacency matrix A_{l+1} for the layer $l+1$ given the adjacency matrices of the previous layers. Precisely, the GFlowNet learns to predict the probability p_i, for $i \in \{1, ..., n\}$, that the node v_i in the neighbourhood of K^l will be included in $\mathcal{V}^{l+1}$, the nodes sampled for layer $l+1$. We use a Bernoulli distribution to define the nodes probability inclusion in the sample. Therefore, the forward probability between layers l and $l+1$ is

$$P_F(s_{l+1}|s_l) = \prod_{v_i \in \mathcal{V}^{l+1}} p_i \prod_{v_i \in \mathcal{N}(K^l) \setminus \mathcal{V}^{l+1}} (1 - p_i) \tag{5}$$

The first product computes the probabilities of the nodes that are included in the new sampled set $\mathcal{V}^{l+1}$, and the second the probabilities of the nodes that are not included.

Backward Probability. Trajectory balance (Equation 3) also requires defining the probability of transitioning backwards through the states. This is however not required in our setup, as the state representation $s = (A_0, ..., A_l)$ exactly saves the trajectory that was taken through $\mathcal{G}_F$ to get to s. We pass this information to the GFlowNet by adding an identifier to the nodes' embeddings indicating in what layer the node was sampled. We also include an identifier for the mini-batch target nodes. This means $\mathcal{G}_F$ is a tree, and has a unique path backwards to the initial state s_0, and so $P_B(s_l|s_{l+1}) = 1$.

Combining our setup with the trajectory balance loss (Equation 3), the GRAPES loss is

$$\mathcal{L}_{GRAPES}(\tau) = \left(\log \frac{Z \prod_{l=1}^{L} P_F(s_l|s_{l-1})}{R(s_L)} \right)^2 \tag{6}$$

$$= \left(\log Z + \sum_{l=1}^{L} \log P_F(s_l|s_{l-1}) + \alpha \mathcal{L}_{GCN_C}(A_0, ..., A_L) \right)^2 \tag{7}$$

We model the normalizer $Z(s_0)$ in Equation 3 as a real-valued trainable parameter $\log Z$ that we update using GRAPES's loss, and so assume $Z(s_0)$ is constant to remove an additional estimation step.

4.2 Reward scaling

In our experiments, we noticed that with the bigger datasets, the GFlowNet is more affected by the log-probabilities than the reward from the classification GCN_C. The reason that the term $\log P_F(s_{l+1}|s_l)$ is dominant in $\mathcal{L}_{GRAPES}$ is that $\log P_F(s_l|s_{l-1}) = \sum_{i=1}^{n} \log p(v_i)$, where n is the size of neighborhood of the nodes sampled in the previous layer l. Given a batch size of 256, this neighborhood can be as big as 52 000 nodes, resulting in summing 52 000 log-probabilities. The majority of the probabilities $p(v_i)$ are values close to zero, therefore, the above sum would be a large negative number. Since the loss $\mathcal{L}_{GCN_C}$ is in our experiments quite close to zero, the log-probability and its variance dominate the loss. Therefore, we add the hyper-parameter α to the reward and tune it in our experiments.

4.3 Sampling and Off-Policy Training

We next describe the sampling technique of GRAPES. We have a clear constraint on the number of nodes we want to include in training: We want to sample exactly k nodes without replacement. However, our forward distribution P_F contains n independent Bernoulli distributions, and it is unlikely that we sample exactly k nodes from this distribution. Instead, we use the Gumbel-max trick (Vieira; Huijben et al., 2022) which selects a set of nodes $\mathcal{V}_k^l$ by perturbing the log probabilities randomly and taking the top-k among those:

$$\mathcal{V}_k^l = \mathsf{top}(k, \log p_1 + G_1, ..., \log p_n + G_n), \quad G_i \sim \text{Gumbel}(0, 1) \tag{8}$$

The sampling process shown above is different from faithfully sampling from the GFlowNet distribution P_F. This means we train the GFlowNet in an off-policy setting. Importantly, GFlowNets, and in particular the trajectory balance loss, can learn from off-policy distributions without adjusting the objective (Malkin et al., 2022b). If we were to use a direct gradient-estimation method, like REINFORCE, we would have to compute the probability of sampling k nodes from P_F for importance weighting, which is expensive (Ahmed et al., 2023). Therefore, the off-policy properties of GFlowNets allow us to use the Gumbel-max trick without having to compute importance weights or any other form of normalization.

4.4 GRAPES ALGORITHM

Algorithm 1 shows one epoch of GRAPES in pseudocode. Note that in layer l, to preserve the self-loops of the nodes sampled in layer $l-1$, we concatenate the nodes sampled in layer $l-1$ to the sampled nodes in layer l, i.e., $K^l = K^{l-1} \cup \mathcal{V}_k^l$. Therefore, we build the adjacency matrix for layer l $A^l \in \mathbb{R}^{|K^{l-1}| \times |K^l|}$ by adding all the edges between K^{l-1} and K^l.

Algorithm 1 One GRAPES epoch

```
Require: Graph $\mathcal{G}_C$, node features $X$, node labels $Y$, target nodes $\mathcal{V}^t$, batch size $B$, sample size $k$,
    GCN for classification $GCN_C$, and GCN for GFlowNet $GCN_F$.
 1: Divide target nodes $\mathcal{V}^t$ into batches $\mathcal{V}^b$ of size B
 2: for each batch $\mathcal{V}^b$ do
 3:     $K^0 = \mathcal{V}^b$
 4:     Build adjacency matrix $A^0$ from $K^0$
 5:     for layer $l = 1$ to $L$ do
 6:         $\mathcal{V}_n^l \leftarrow \mathcal{N}(K^{l-1})$                                  ▷ Get all $n$ neighbors of $K^{l-1}$
 7:         $p_1, ..., p_n \leftarrow GCN_F(A_0, ..., A_{l-1})$                  ▷ Compute probabilities of inclusion
 8:         $G_1, ..., G_n \sim \text{Gumbel}(0, 1)$                              ▷ Sample Gumbel noise
 9:         $\mathcal{V}_k^l \leftarrow \mathsf{top}(k, \log p_1 + G_1, ..., \log p_n + G_n)$      ▷ Get $k$ best nodes (Eq. 8)
10:         $K^l = K^{l-1} \cup \mathcal{V}_k^l$                                   ▷ Add new nodes
11:         Build adjacency matrix $A^l$ from $K^l$
12:     end for
13:     Pass all $\hat{A}_l$ to $GCN_C$ and obtain classification loss $\mathcal{L}_{GCN_C}$ given $\{(X_j, Y_j)\}_{j=1}^B$
14:     $R(A_0, ..., A_L) \leftarrow \exp(-\alpha \cdot \mathcal{L}_{GCN_C})$                   ▷ Calculate reward
15:     Compute GRAPES loss $\mathcal{L}_{GRAPES}$ from Eq. 6
16:     Update parameters of $GCN_F$ by minimizing $\mathcal{L}_{GRAPES}$
17:     Update parameters of $GCN_C$ by minimizing $\mathcal{L}_{GCN_C}$
18: end for
```

5 EXPERIMENTS

In this section, we evaluate GRAPES on several graphs of varying scales, and compare it with the existing sampling methods.

5.1 EXPERIMENTAL SETUP

For all our experiments, we use a two-layer GCN with ReLU activation function for both classification and GFlowNet networks, GCN_C and GCN_F. We evaluate GRAPES on the following datasets for node classification task: citation networks (Cora, Citeseer, Pubmed with the "full" split) (Sen et al., 2008; Yang et al., 2016), Reddit (Hamilton et al., 2017), Flickr (Zeng et al., 2019), Yelp (Zeng et al., 2019), and two open graph benchmarks ogbn-arxiv and ogbn-products (Hu et al., 2020). Statistics about the datasets can be found in Appendix B. We conduct our experiments on a machine with an Nvidia RTX A6000 48 GB GPU.

Table 1: F1-scores (%) for different sampling methods for a batch size of 256, and 256 samples. We report the mean and standard deviation over 10 runs. The best values are in **bold**, and the second best are $\underline{\text{underlined}}$. OOM stands for "out-of-memory", which signifies that the sampling process required more GPU memory than initially allocated.

Method	**Cora**	**Citeseer**	**Pubmed**	**Reddit**	**Flickr**
FastGCN	74.93 ± 6.62	53.26 ± 12.39	53.73 ± 23.05	38.02 ± 6.26	41.06 ± 6.50
LADIES	69.90 ± 18.43	67.85 ± 8.43	56.36 ± 25.29	81.88 ± 3.87	43.23 ± 0.73
GraphSAINT	86.20 ± 0.85	77.63 ± 0.54	83.07 ± 0.89	80.50 ± 1.00	**48.47 ± 1.05**
AS-GCN	$\underline{86.42}$ ± 0.31	**78.80 ± 0.21**	$\underline{89.76}$ ± 0.53	$\underline{92.44}$ ± 0.56	$\underline{48.21}$ ± 1.70
GRAPES (ours)	**87.29 ± 0.15**	$\underline{78.74}$ ± 0.18	**90.11 ± 0.24**	**93.68 ± 0.06**	47.33 ± 1.41

	Yelp	**ogbn-arxiv**	**ogbn-products**	**Average F1**	**Average Rank**
FastGCN	26.42 ± 2.05	29.91 ± 3.57	33.01 ± 4.03	43.79	4.625*
LADIES	16.02 ± 2.28	44.58 ± 8.12	$\underline{67.72}$ ± 1.95	55.94	3.875*
GraphSAINT	33.13 ± 0.94	53.71 ± 2.79	59.57 ± 0.65	65.29	2.750
AS-GCN	$\underline{30.99}$ ± 1.21	**65.95 ± 1.91**	OOM	$\underline{70.37}$	$\underline{2.250}$
GRAPES (ours)	**44.91 ± 0.08**	$\underline{64.54}$ ± 0.51	**73.65 ± 0.21**	**72.53**	**1.500**

5.2 BASELINES EVALUATION PROTOCOL

We compare GRAPES with the following baselines: FastGCN (Chen et al., 2018b), LADIES (Zou et al., 2019), GraphSAINT (Zeng et al., 2019), GAS (Fey et al., 2021), and AS-GCN (Huang et al., 2018). Our evaluation protocol consists of two steps: 1) training a GCN classifier via sampling and 2) evaluating the classifier's performance on the test set using the full graph. We implement GRAPES and GraphSAINT using PyTorch Geometric (Fey & Lenssen, 2019) and refer to the source code of the original paper for the remaining baselines. The original papers evaluated the baseline methods under different configurations of GCN classifier architecture, data pre-processing, and regularization. To provide a fair comparison and study the behaviour of the sampling methods in isolation, we modify the GCN architecture of the baselines so that all the methods, including GRAPES, have the same two-layer classification GCN architecture with a fixed hidden dimension.

For selecting the learning rate, we refer to the original papers for the baselines. GRAPES requires selecting two additional hyperparameters: the learning rate for the GFlowNet and the scaling parameter α. We tune these based on classification performance on the validation set. These hyperparameters are specific to our sampling method and do not explicitly affect the learning capacity of the GCN classifier. We refer the reader to Appendix A for details on hyperparameter settings.

5.3 RESULTS

Comparison with sampling methods We present the F1 scores of GCNs trained via sampling for GRAPES and the sampling-based baselines in Table 1. We observe that GRAPES often outperforms all these baselines. Unlike the other methods, GRAPES ranks consistently within the top 3 methods, and its average rank is the highest, suggesting that its ability to adapt the sampling policy to particular features of the data allows GRAPES to maintain consistent performance across datasets. We probe into the observed differences in rank by performing two statistical tests, using the ranks obtained for each sampling method over all datasets. First, a Friedman test reveals significant differences in ranks between all methods. We subsequently apply an all-pairs comparison and find that GRAPES ranks significantly higher than FastGCN and LADIES. AS-GCN, the next high-rank method, only significantly differs from FastGCN. The Appendix C provides more detail on our methodology for hypothesis testing.

An additional desirable property we observe is the low variance in the performance of GRAPES. After executing each experiment 10 times, we note that GRAPES has the lowest standard deviation, except for two datasets, unlike FastGCN and LADIES, which exhibit high variance. Our results demonstrate that GRAPES offers robust performance, maintaining consistency over different datasets and the randomness involved in multiple runs. FastGCN and LADIES, which use a fixed policy, fail to compete with the other methods in our experimental setup. They assign a higher probability to

Table 2: Comparison of F1-Scores (%) for GAS, and GRAPES-32 and GRAPES-256 which correspond to the different sampling sizes. We report the mean and standard deviation for 10 repeats. The best values are in **bold**.

	Cora	**Citeseer**	**Pubmed**	**Reddit**
GAS	87.67 ± 0.21	**79.37** ± **0.35**	87.94 ± 0.31	**94.83** ± **0.83**
GRAPES-32	**88.10** ±**0.14**	79.04 ± 0.17	89.58 ± 0.17	92.43 ±0.13
GRAPES-256	87.29 ± 0.15	78.74 ± 0.18	**90.11** ± **0.24**	93.68 ± 0.06
	Flickr	**Yelp**	**ogbn-arxiv**	**ogbn-products**
GAS	**51.32** ± **0.30**	33.79 ± 0.61	**69.38** ± **0.21**	**75.12** ± **0.17**
GRAPES-32	45.69 ±1.33	44.50 ±0.26	64.03 ±0.57	73.62 ±0.16
GRAPES-256	47.33 ± 1.41	**44.91** ± **0.08**	64.54 ± 0.51	73.65 ± 0.21

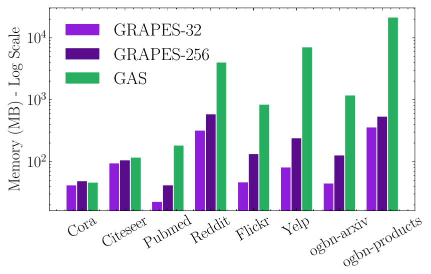

Figure 2: GPU peak memory allocation (MB) for GAS, and GRAPES-32 and GRAPES-256.

nodes with higher degrees. This heuristic results in neglecting informative low-degree nodes. As the results show, GraphSAINT fails to outperform the more competitive baselines.

AS-GCN is the only adaptive sampling method among the baselines and is the next best-performing method in this category, indicating the importance of adaptive sampling. However, unlike GRAPES, this method is only adaptive to the sampling variance. Furthermore, it uses a large amount of memory because it utilizes an attention mechanism during sampling. Therefore, AS-GCN causes out-of-memory errors when training on the largest dataset, ogbn-products, indicating AS-GCN's failure to scale up to large graphs.

Comparison with a non-sampling scalable method GAS (Fey et al., 2021) is a non-sampling method that uses the historical embeddings of the 1-hop neighbors of the target nodes. For GRAPES, we report the values for sampling sizes 32 and 256. We present classification results in Table 2, and we visualize memory usage in Fig. 2. While GAS has higher classification F1-scores for some datasets, GRAPES achieves comparable F1-scores and significantly outperforms GAS for Yelp. The memory usage results show that even with a large sample size, GRAPES can use up to an order of magnitude less GPU memory than GAS, especially for large datasets. In large, densely connected graphs such as Reddit, the 1-hop neighborhood can be very big. Then, the difference in memory use for GRAPES, which only sees a small set of neighbors, and GAS, which uses all the neighbors, is noticeable. While GAS occasionally achieves higher F1-scores, it consistently demands more memory, indicating a potential compromise between accuracy and computational efficiency. In contrast, both variants of GRAPES variants strike a balance, delivering comparable F1 scores with more modest memory footprints.

GRAPES is robust to low sample sizes. Figure 3 shows the effects of changing the sample size for all the methods except for GAS, which does not use any sampling and learns from a fixed number of nodes. To quantify the robustness to changes in the sample size, we measure deviations from the mean F1-score and find that GRAPES has the smallest deviation from the horizontal line of best fit (see Appendix D for details). This shows that the classification accuracy of GRAPES is robust to the sample size. Therefore, GRAPES achieves strong performance with fewer sampled nodes needed than the baselines, enabling training GCNs on larger graphs while using less GPU memory.

GRAPES learns strong preferences over nodes. The ability to selectively choose *influential* nodes is a crucial property of GRAPES. Figure 4 shows the mean and standard deviation of base 2 entropy for the node preference probabilities for the two layers of GCN_C for ogbn-products. The probabilities show preference towards particular nodes with a Bernoulli distribution. A well-trained model must have a high preference (probability close to 1) for some nodes and a low preference (probability close to 0) for the rest. Therefore, we would like a low average entropy with a high standard deviation. As the figure shows, the mean entropy in both layers decreases from almost 1 and converges to a value above 0 while the standard deviation increases. This indicates that for ogbn-products, the GFlowNet initially assigned a probability near 0.5, indicating little preference. However, after several training epochs, GRAPES starts preferring some nodes, resulting in lower mean entropy. We observe similar behavior for the large graphs (Reddit, Yelp, ogbn-arxiv, and ogbn-products). However, we observe that the GFlowNet exhibits no preferences among the nodes for the other datasets. For more details about the rest of the datasets, see Appendix E.

6 Discussion and Conclusion

We propose GRAPES, a GFlowNet-based graph sampling method that facilitates the scalability of training GNNs on very large graphs. GRAPES learns the importance of the nodes by adapting to the classification loss. We show how a GFlowNet can help build a subgraph of influential nodes by learning the node preferences adaptive to the node features, GNN architecture, classification task, and graph topology. Our experiments demonstrate that GRAPES effectively selects nodes from large-scale graphs and achieves consistent performance over state-of-the-art sampling methods. Moreover, we show GRAPES compared to the other sampling methods, can maintain a high level of classification accuracy even with lower sampling rates, indicating GRAPES' ability to scale to larger graphs by sampling a small but influential set of nodes. GRAPES achieves comparable performance to GAS, while using up to an order of magnitude less memory.

Lack of Uniform Evaluation Protocol. Existing methods in the literature report performance on graph sampling under settings with different GCN architectures, regularization techniques, feature normalization strategies, and data splits, among other differences. These differences made it challenging to determine the benefits of each sampling method. This motivated our decision to implement a unified protocol across all methods where we keep the architecture fixed. We encourage future work to consider a similar methodology for a fair evaluation of sampling methods, or an experimental review study, as is common in other areas of machine learning research on graphs (Shchur et al., 2018; Ruffinelli et al., 2019).

Known Limitations. In our experiments, we focused on the problem of node classification. However, GRAPES is not tied to a particular downstream task. In general, GFlowNets assume access to a tractable reward function (Bengio et al., 2021b). Therefore, GRAPES will be applicable for other graph-related tasks with tractable rewards, like link prediction and unsupervised representation learning. We leave these directions for future work. Furthermore, we assumed that the normalizing function $Z(s_0)$ in Equation 3 is constant across different mini-batches to reduce estimation overhead, and we leave the study on the effect of estimating $Z(s_0)$ on the sampling for future work.

Reproducibility Statement. Our code is publicly available at `https://github.com/dfdazac/grapes`. The full set of hyperparameters used for GRAPES are available in the repository. As described previously, we adapted the baselines (LADIES, GAS, AS-GCN, FastGCN, GraphSAINT) for a fair comparison; the link to the code and the corresponding hyperparameters for the modified baseline models can be found in our repository. The link to the code and data for the data analyses we carried out are also available in our repository.

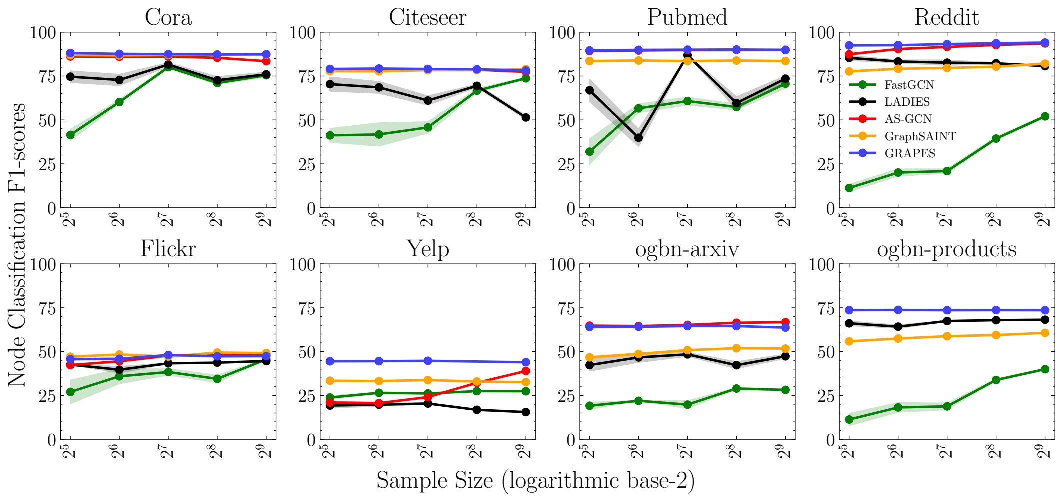

Figure 3: Comparative analysis of classification accuracy across different sampling sizes for sampling baseline and GRAPES. We repeated each experiment five times: The shaded regions show the 95% confidence intervals. AS-GCN is missing for ogbn-products due to OOM errors.

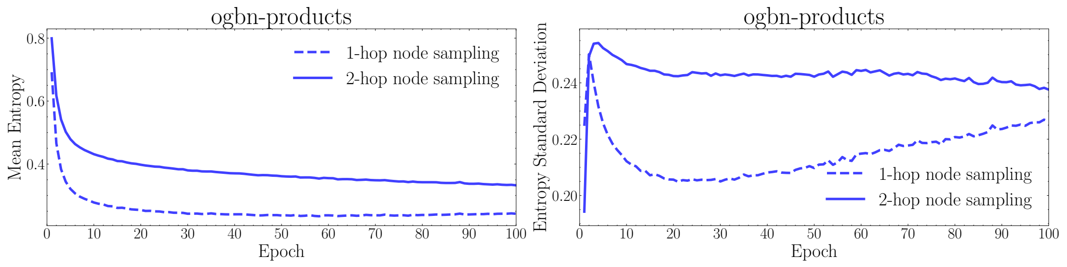

Figure 4: Entropy for the ogbn-products dataset. **Left:** Comparison of average mean entropy values over epochs. **Right:** Comparison of standard deviation entropy values over epochs.

Acknowledgements

This project was partially funded by Elsevier's Discovery Lab and Huawei DREAMS Lab. We thank Michael Cochez and Ruud van Bakel for insightful discussions.

References

Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *international conference on machine learning*, pp. 21–29. PMLR, 2019.

Kareem Ahmed, Zhe Zeng, Mathias Niepert, and Guy Van den Broeck. SIMPLE: A gradient estimator for k-subset sampling. In *The Eleventh International Conference on Learning Representations*, 2023.

Emmanuel Bengio, Moksh Jain, Maksym Korablyov, Doina Precup, and Yoshua Bengio. Flow network based generative models for non-iterative diverse candidate generation. *Advances in Neural Information Processing Systems*, 34:27381–27394, 2021a.

Yoshua Bengio, Salem Lahlou, Tristan Deleu, Edward J Hu, Mo Tiwari, and Emmanuel Bengio. Gflownet foundations. *arXiv preprint arXiv:2111.09266*, 2021b.

Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. In *International Conference on Machine Learning*, pp. 942–950. PMLR, 2018a.

Jie Chen, Tengfei Ma, and Cao Xiao. FastGCN: Fast learning with graph convolutional networks via importance sampling. In *6th International Conference on Learning Representations, ICLR 2018, Conference Track Proceedings*. OpenReview.net, 2018b.

Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui 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*, pp. 257–266, 2019.

Tristan Deleu, António Góis, Chris Emezue, Mansi Rankawat, Simon Lacoste-Julien, Stefan Bauer, and Yoshua Bengio. Bayesian structure learning with generative flow networks. In *Uncertainty in Artificial Intelligence*, pp. 518–528. PMLR, 2022.

Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine learning research*, 7:1–30, 2006.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

Matthias Fey, Jan E Lenssen, Frank Weichert, and Jure Leskovec. Gnnautoscale: Scalable and expressive graph neural networks via historical embeddings. In *International conference on machine learning*, pp. 3294–3304. PMLR, 2021.

Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*, 32(200):675–701, 1937.

Wenhao Gao, Tianfan Fu, Jimeng Sun, and Connor Coley. Sample efficiency matters: a benchmark for practical molecular optimization. *Advances in Neural Information Processing Systems*, 35: 21342–21357, 2022.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.

Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. Adaptive sampling towards fast graph representation learning. *Advances in neural information processing systems*, 31, 2018.

Iris AM Huijben, Wouter Kool, Max B Paulus, and Ruud JG Van Sloun. A review of the gumbel-max trick and its extensions for discrete stochasticity in machine learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(2):1353–1371, 2022.

Moksh Jain, Emmanuel Bengio, Alex Hernandez-Garcia, Jarrid Rector-Brooks, Bonaventure FP Dossou, Chanakya Ajit Ekbote, Jie Fu, Tianyu Zhang, Michael Kilgour, Dinghuai Zhang, et al. Biological sequence design with gflownets. In *International Conference on Machine Learning*, pp. 9786–9801. PMLR, 2022.

Moksh Jain, Tristan Deleu, Jason Hartford, Cheng-Hao Liu, Alex Hernandez-Garcia, and Yoshua Bengio. Gflownets for AI-driven scientific discovery. *Digital Discovery*, 2(3):557–577, 2023.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

Michelle M Li, Kexin Huang, and Marinka Zitnik. Graph representation learning in biomedicine and healthcare. *Nature Biomedical Engineering*, 6(12):1353–1369, 2022.

Wenqian Li, Yinchuan Li, Zhigang Li, Jianye Hao, and Yan Pang. Dag matters! gflownets enhanced explainer for graph neural networks. *arXiv preprint arXiv:2303.02448*, 2023.

Ziqi Liu, Zhengwei Wu, Zhiqiang Zhang, Jun Zhou, Shuang Yang, Le Song, and Yuan Qi. Bandit samplers for training graph neural networks. *Advances in Neural Information Processing Systems*, 33:6878–6888, 2020.

Nikolay Malkin, Moksh Jain, Emmanuel Bengio, Chen Sun, and Yoshua Bengio. Trajectory balance: Improved credit assignment in gflownets. *Advances in Neural Information Processing Systems*, 35: 5955–5967, 2022a.

Nikolay Malkin, Salem Lahlou, Tristan Deleu, Xu Ji, Edward Hu, Katie Everett, Dinghuai Zhang, and Yoshua Bengio. Gflownets and variational inference. *arXiv preprint arXiv:2210.00580*, 2022b.

David F Nettleton. Data mining of social networks represented as graphs. *Computer Science Review*, 7:1–34, 2013.

Karl Pearson. X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 50(302):157–175, 1900.

Daniel Ruffinelli, Samuel Broscheit, and Rainer Gemulla. You can teach an old dog new tricks! on training knowledge graph embeddings. In *International Conference on Learning Representations*, 2019.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

Zhihao Shi, Xize Liang, and Jie Wang. Lmc: Fast training of gnns via subgraph sampling with provable convergence. *arXiv preprint arXiv:2302.00924*, 2023.

Maksim Terpilowski. scikit-posthocs: Pairwise multiple comparison tests in python. *The Journal of Open Source Software*, 4(36):1169, 2019. doi: 10.21105/joss.01169.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio, et al. Graph attention networks. *stat*, 1050(20):10–48550, 2017.

T. Vieira. Gumbel-max trick and weighted reservoir sampling. URL `http://timvieira.github.io/blog/post/2014/08/01/gumbel-max-trick-and-weighted-reservoir-sampling/`.

Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. Graph neural networks in recommender systems: a survey. *ACM Computing Surveys*, 55(5):1–37, 2022.

Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pp. 40–48. PMLR, 2016.

Minji Yoon, Théophile Gervet, Baoxu Shi, Sufeng Niu, Qi He, and Jaewon Yang. Performance-adaptive sampling strategy towards fast and accurate graph neural networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 2046–2056, 2021.

Haiyang Yu, Limei Wang, Bokun Wang, Meng Liu, Tianbao Yang, and Shuiwang Ji. Graphfm: Improving large-scale gnn training via feature momentum. In *International Conference on Machine Learning*, pp. 25684–25701. PMLR, 2022.

Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. Graph transformer networks. *Advances in neural information processing systems*, 32, 2019.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. GraphSAINT: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931*, 2019.

Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. Layer-dependent importance sampling for training deep and large graph convolutional networks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019*, pp. 11247–11256, 2019.

A Experimental Details

Our experiments were carried out in a single-node cluster setup. We conducted our experiments on a machine with an Nvidia RTX A6000 GPU (48GB GPU memory) and each machine had 48 CPUs.

A.1 Hyperparameter Tuning

We tune the hyperparameters of GRAPES using a random search strategy with the goal of maximizing the accuracy of the validation dataset. We used Weights and Biases for hyperparameter tuning [4]. The best-performing hyperparameters for every dataset can be found in our repository `https://github.com/dfdazac/grapes`. The following are the hyperparameters that we tuned: the learning rate of the GFlowNet, the learning rate of the classification GCN, and the scaling parameter α. We used the log uniform distribution to sample the aforementioned hyperparameters with the values from the following ranges respectively, $[1e-6, 1e-2]$, $[1e-6, 1e-2]$, and [1e2, 1e6]. We kept the other hyperparameters such as the batch size and hidden dimension of the GCN. We used the Adam optimizer (Kingma & Ba, 2014) for GCN_C and GCN_F. The number of epochs was selected between 50, 100, and 150 depending on the performance on the validation set.

A.2 Baselines

For a fair comparison, we adjusted the implementations of the baselines so that the only difference is the sampling methods and the rest of the training conditions are kept the same. In the following, we explain the details of the modifications to each of the baselines.

For LADIES we used the official implementation, which also contains an implementation of FastGCN. We changed the nonlinear activation function from ELU to ReLU, and we removed any linear layers after the two layers of the GCN, set dropout to zero, and disabled early stopping. We also noticed that the original LADIES implementation divided the target nodes into mini-batches, not from the entire graphs as we do, but into random fragments. This means that LADIES and FastGCN do not see all of the target nodes in the training data. We kept this setting unchanged, because otherwise it significantly slowed down the training of these two methods.

For GraphSAINT we noticed that the GNN consists of two layers of higher order aggregator, which is a combination of GraphSage-mean (Hamilton et al., 2017) and MixHop (Abu-El-Haija et al., 2019), and a linear classification layer in the end. Moreover, the original implementation of GraphSAINT is only applicable to inductive learning on the graphs, where the train graph only contains the train nodes and is entirely different from the validation and test graphs where only the nodes from the validation set and test are available respectively. We argue that in transductive learning unlike inductive learning, the motivation to scale to larger graphs is higher since the validation and test nodes are also available during training, and therefore, the processed graph is larger. Finally we used Pytorch Geometric's function for GraphSAINT node sampling to keep all the configurations the same as GRAPES. Therefore, we use the same GCN and data loader (transductive) as ours and use GraphSAINT to sample a subgraph of nodes for training. We use the node sampler setting since it is the only setting that allows specifying different sampling budgets, and therefore, can be compared to layer-wise methods with the same sampling budget. We also removed the early stopping and use the same number of epochs as GRAPES. Please refer to our repository for more details about the implementation of GraphSAINT.

For GAS, we used the original implementation. However, we changed the configuration of the GCN to have a two-layer GCN with 256 hidden units. We turned off dropout, batch normalisation and residual connections in the GCN. We also removed early stopping for the training.

For AS-GCN we removed the attention mechanism used in the GCN classifier. Their method also uses attention in the sampler, which is separate from the classifier, so we keep it.

[4] `https://wandb.ai`

Table 3: Statistics of the datasets used in our experiments. The label rate indicates the percentage of nodes used for training.

Dataset	Task	Nodes	Edges	Features	Classes	Label Rate (%)
Cora	multi-class	2,708	5,278	1,433	7	44.61
CiteSeer	multi-class	3,327	4,552	3,703	6	54.91
PubMed	multi-class	19,717	44,324	500	3	92.39
Reddit	multi-class	232,965	11,606,919	602	41	65.86
Flickr	multi-class	89,250	449,878	500	7	50.00
Yelp	multi-label	716,847	6,977,409	300	100	75.00
ogbn-arxiv	multi-class	169,343	1,157,799	128	40	53.70
ogbn-products	multi-class	2,449,029	61,859,076	100	47	8.03

B Dataset statistics

We present the statistics of the datasets used in our experiments in Table 3. The splits that we used for Cora, Citeseer, and Pubmed correspond to the "full" splits, in which the label rate is higher than in the "public" splits.

C Methodology for hypothesis testing

In section 5.2 we present the classification performance of a GCN classifier, after training it with the sampling baselines and GRAPES. We then perform statistical tests aimed at answering two questions.

Are the differences between ranks statistically significant? Our null hypothesis is that the ranks of all methods have the same distribution. We present the ranks in Table 4, together with the resulting average rank. We apply a Friedman test (Friedman, 1937) to compare the ranks of each method, obtaining a p-value of 0.0005. We thus reject the null hypothesis and conclude that the observed difference in rank across methods is statistically significant.

Is the difference in ranks between specific pairs of methods statistically significant? In this case, we are interested in multiple null hypotheses involving pairs of methods. For example, *do the ranks obtained from GRAPES and FastGCN come from the same distribution?* As we are interested in similar hypotheses involving different methods, this requires comparing multiple pairs and adjusting the p-value to account for multiple comparisons (Demšar, 2006). We carry out a post-hoc Nemenyi's test using the `scikit-posthocs` library (Terpilowski, 2019) and obtain the p-values shown in Table 5. The results show that the difference in rank between GRAPES is significant with respect to FastGCN and LADIES with $p < 0.05$.

D Pearson's Chi-squared (χ^2) test as a Robustness Measure

Figure 3 shows the effect of changing sample size on the node classification performance. In an ideal case, it would be desirable to have a constant accuracy as the number of the sample size changes. To quantify the robustness of the different sampling methods, we compute the goodness-of-fit of

Table 4: Ranks according to the F1 scores obtained by each sampling method on the node classification task. Asterisks (*) indicate that the difference in average rank to GRAPES is statistically significant at $p < 0.05$.

Method	Cora	Citeseer	Pubmed	Reddit	Flickr	Yelp	ogbn-arxiv	ogbn-products	Average
FastGCN	4	5	5	5	5	4	5	4	4.625*
LADIES	5	4	4	3	4	5	4	2	3.875*
GraphSAINT	3	3	3	4	1	2	3	3	2.750
AS-GCN	2	1	2	2	2	3	1	5	2.250
GRAPES (ours)	1	2	1	1	3	1	2	1	**1.500**

Table 5: p values obtained with the Nemenyi post-hoc test for comparing the ranks for all pairs of methods.

Method	FastGCN	LADIES	GraphSAINT	AS-GCN	GRAPES
FastGCN	1.0000	0.8669	0.1233	0.0223	0.0010
LADIES	0.8669	1.0000	0.5977	0.2396	0.0223
GraphSAINT	0.1233	0.5977	1.0000	0.9000	0.5080
AS-GCN	0.0223	0.2396	0.9000	1.0000	0.8669
GRAPES (ours)	0.0010	0.0223	0.5080	0.8669	1.0000

Table 6: Chi-square values. Each experiment was repeated five times. The χ^2 were computed with the mean accuracies. The best values are in **bold**, and the second best are <u>underlined</u>. OOM indicates an out-of-memory error. Asterisks (*) indicate that the difference in average rank to GRAPES is statistically significant at $p < 0.05$.

Method	**Cora**	**Citeseer**	**Pubmed**	**Reddit**	**Flickr**
FastGCN	14.4540	17.3168	14.7477	38.3733	4.9597
LADIES	0.7089	4.0147	18.6169	0.1393	0.3329
GraphSAINT	**0.0016**	**0.0130**	**0.0013**	<u>0.1304</u>	0.1004
AS-GCN	0.0602	<u>0.0137</u>	0.0033	0.2550	0.6420
GRAPES	<u>0.0047</u>	0.0616	<u>0.0018</u>	**0.0209**	**0.0911**
	Yelp	**ogbn-arxiv**	**ogbn-products**	**Average χ^2**	**Average Rank**
FastGCN	0.3335	3.6756	23.4954	14.6695	4.6250*
LADIES	0.9420	0.7474	<u>0.1607</u>	3.2079	3.7500*
GraphSAINT	<u>0.0232</u>	0.3967	0.2358	<u>0.1128</u>	2.0000
AS-GCN	9.2825	<u>0.0599</u>	OOM	1.4738	3.0000
GRAPES	**0.0084**	**0.0085**	**0.0002**	**0.0246**	1.6250

observed accuracy values to a horizontal line of best fit (*i.e.* a constant accuracy value). To this end, we use Pearson's chi-squared test (Pearson, 1900). Let's assume we have a set of observed accuracies, denoted as O_i, and a horizontal line of best fit which represents the expected accuracy, E_i. The chi-square statistic, χ^2, is computed using the formula:

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}, \tag{9}$$

where n is the number of observed values. A high χ^2 value would indicate that the observed accuracies significantly deviate from the expected accuracy, while a low value would suggest that they are close. The expected accuracy E_i is the average accuracy across all observations for a particular combination of method and dataset. We did not include GAS (Fey et al., 2021) as the approach does not sample any nodes.

Table 6 shows the χ^2 values for the different sampling methods we study. GRAPES has the lowest average χ^2 across all the different datasets, which implies that is very robust to changes in the sampling sizes compared to the baseline sampling methods.

Rank Significance Test. Our null hypothesis is that the ranks of all methods have the same distribution. We also performed a Friedman test to compare the ranks of each method, obtaining *p*-value of 0.000627. Therefore, we reject null hypothesis, and conclude that the ranks presented in Table 6 are statistically significant.

E Entropy as Node preference measure

Figures 5 and 6 show the mean and standard deviation of entropy in base two of all the datasets. We calculate the mean entropy for as the following:

$$E = \frac{1}{n}\sum_{i=1}^{n} p_i \cdot \log_2(p_i) + (1 - p_i) \cdot \log_2(1 - p_i) \tag{10}$$

where n is the number of neighbors of the nodes sampled in the previous layer and p_i is the probability of inclusion for each node, which is the output of the GFlowNet. As the figures show, for the small datasets (Cora, Citeseer, Pubmed, Flickr) the mean entropy is: 1) very close to 1, indicating that GRAPES prefers every nodes with the probability close to 0.5, or 2) close to 0 but also with a low standard deviation, meaning that it equally prefers the majority of the nodes with the probability 1 or 0. On the contrary, for the large datasets (Reddit, Yelp, ogbn-arxiv, ogbn-products) by the end of training, the average entropy is lower than 1, with a standard deviation around 0.3 indicating that GRAPES learns different preferences over different nodes, some with a probability close to 1, and some close to 0.

F GPU Memory usage comparison between GRAPES and GAS

We compared different variants of GRAPES, with different sample sizes (32, 256), with GAS (Fey et al., 2021), which is a non-sampling method. Figure 2 shows the GPU memory allocation (MB), in a logarithmic scale, and the F1-scores for GRAPES-32, GRAPES-256 and GAS. The three graph methods exhibit distinct performance characteristics across various datasets. We used the `max_memory_allocated` function in PyTorch to measure the GPU memory allocation.[5] Since this function measures the maximum memory allocation since the beginning of the program, where the memory measurement is done does not matter.

[5] https://pytorch.org/docs/stable/generated/torch.cuda.max_memory_allocated.html

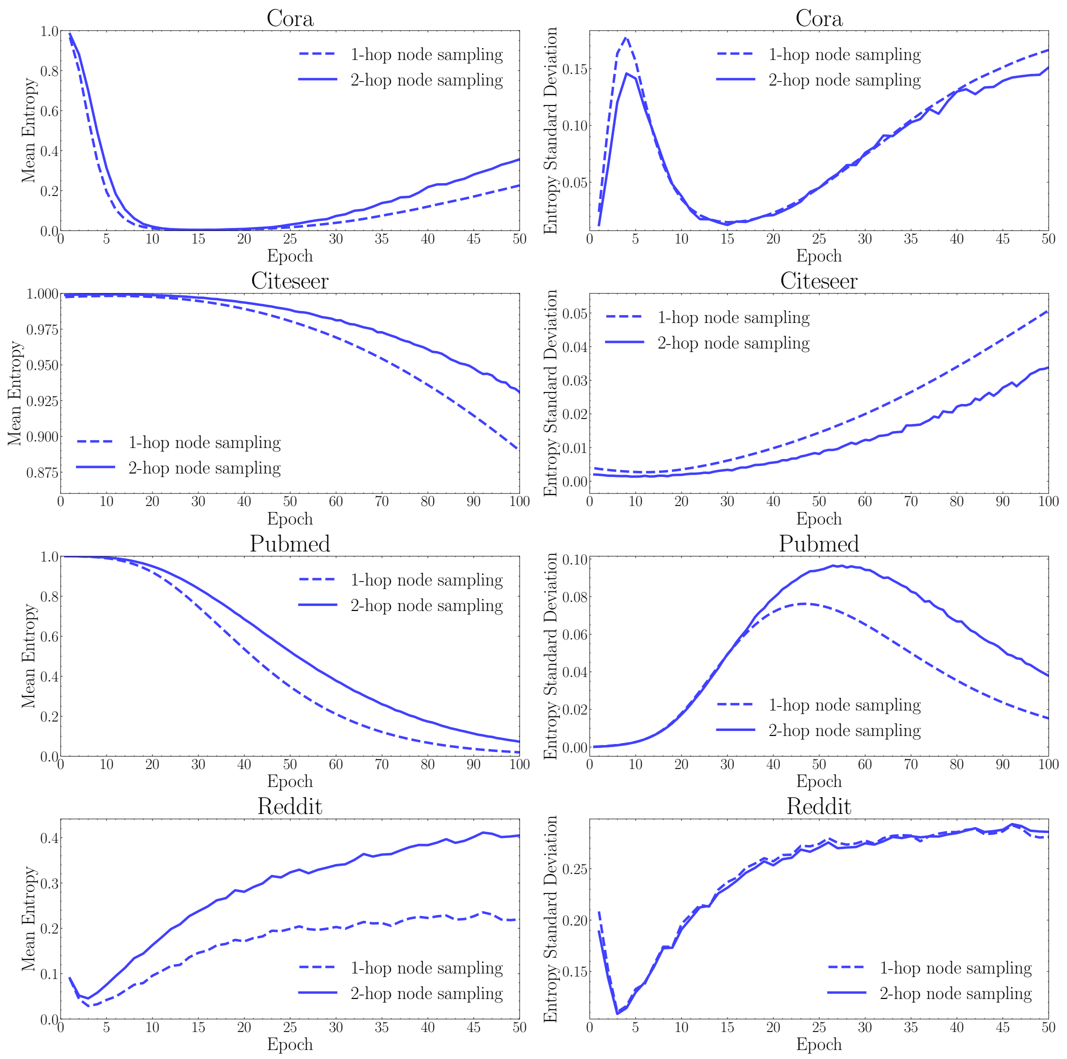

Figure 5: Combined entropy plots for Citeseer, Cora, Flickr and ogbn-arxiv, showcasing the mean entropy and entropy standard deviation across epochs. The plots compare 1-hop node sampling against 2-hop node sampling.

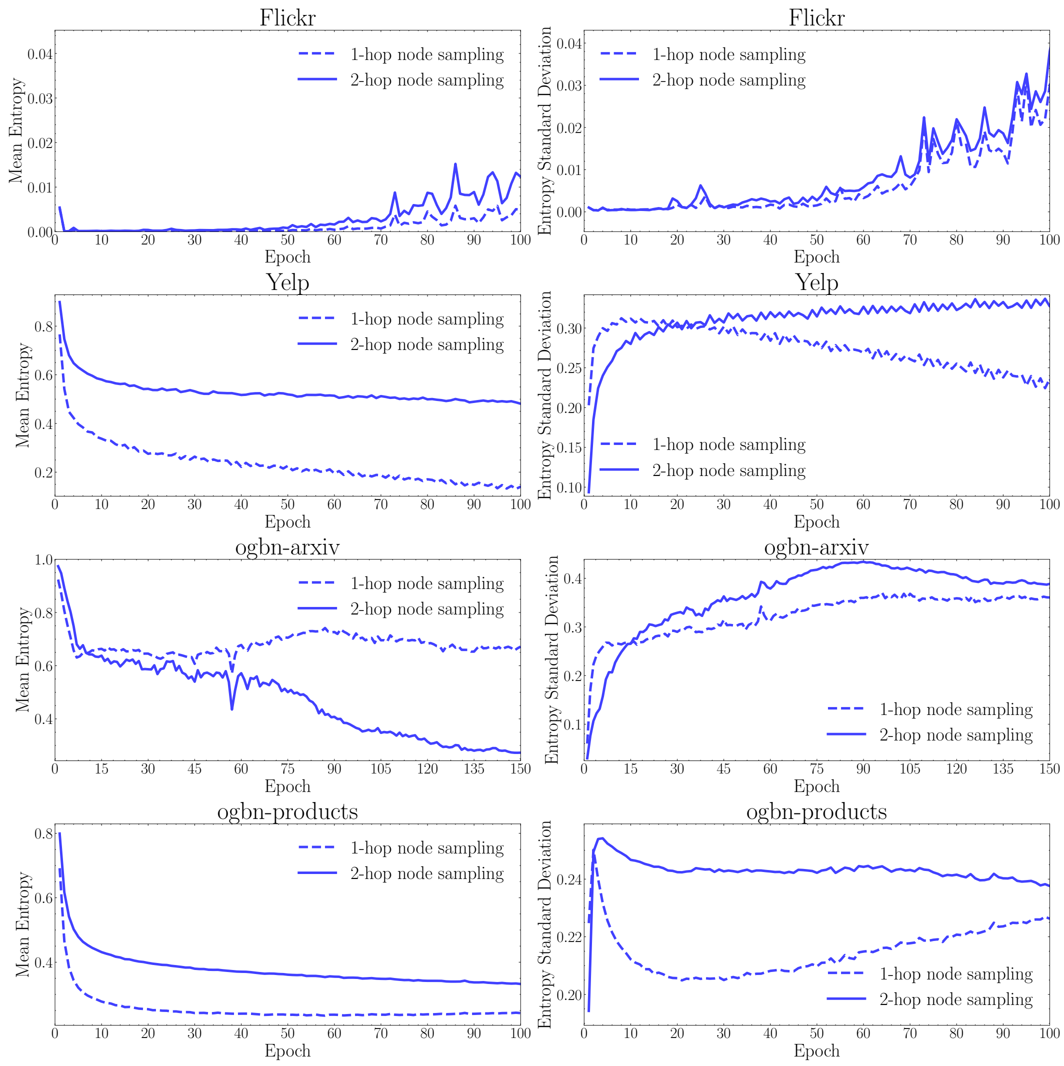

Figure 6: Combined entropy plots for ogbn-products, Pubmed, Reddit and Yelp, showcasing the mean entropy and entropy standard deviation across epochs. The plots compare 1-hop node sampling against 2-hop node sampling.