FLAG: ADVERSARIAL DATA AUGMENTATION FOR GRAPH NEURAL NETWORKS

Kezhi Kong[1], **Guohao Li**[2], **Mucong Ding**[1], **Zuxuan Wu**[1], **Chen Zhu**[1],
Bernard Ghanem[2], **Gavin Taylor**[3], **Tom Goldstein**[1]
[1]University of Maryland, [2]KAUST, [3]US Naval Academy
{kong,mcding,zxwu,chenzhu,tomg}@cs.umd.edu,taylor@usna.edu,
{guohao.li,bernard.ghanem}@kaust.edu.sa

ABSTRACT

Data augmentation helps neural networks generalize better, but it remains an open question how to effectively augment graph data to enhance the performance of GNNs (Graph Neural Networks). While most existing graph regularizers focus on augmenting graph topological structures by adding/removing edges, we offer a novel direction to augment in the input node feature space for better performance. We propose a simple but effective solution, FLAG (Free Large-scale Adversarial Augmentation on Graphs), which iteratively augments node features with gradient-based adversarial perturbations during training, and boosts performance at test time. Empirically, FLAG can be easily implemented with a dozen lines of code and is flexible enough to function with any GNN backbone, on a wide variety of large-scale datasets, and in both transductive and inductive settings. Without modifying a model's architecture or training setup, FLAG yields a consistent and salient performance boost across both node and graph classification tasks. Using FLAG, we reach state-of-the-art performance on the large-scale `ogbg-molpcba`, `ogbg-ppa`, and `ogbg-code` datasets. We open source our implementation at `https://github.com/devnkong/FLAG`.

1 INTRODUCTION

Graph Neural Networks (GNNs) have emerged as powerful architectures for learning and analyzing graph representations. The Graph Convolutional Network (GCN) (Kipf & Welling, 2016) and its variants have been applied to a wide range of tasks, including visual recognition (Zhao et al., 2019; Shen et al., 2018), meta-learning (Garcia & Bruna, 2017), social analysis (Qiu et al., 2018; Li & Goldwasser, 2019), and recommender systems (Ying et al., 2018). However, the training of GNNs on large-scale datasets usually suffers from overfitting, and realistic graph datasets often involve a high volume of out-of-distribution test nodes (Hu et al., 2020), posing significant challenges for prediction problems.

One promising solution to combat overfitting in deep neural networks is data augmentation (Krizhevsky et al., 2012), which is commonplace in computer vision tasks. Data augmentations apply label-preserving transformations to images, such as translations and reflections. As a result, data augmentation effectively enlarges the training set while incurring negligible computational overhead. However, it remains an open problem how to effectively generalize the notion of data augmentation to GNNs. Transformations on images rely heavily on image structures, and it is challenging to design low-cost transformations that preserve semantic meaning for non-visual tasks like natural language processing (Wei & Zou, 2019) and graph learning. Generally speaking, graph data for machine learning comes with graph structure (or edge features) and node features. In the limited cases where data augmentation can be done on graphs, it generally focuses exclusively on the graph structure by adding/removing edges (Rong et al., 2019). To date, there is no study on how to manipulate graphs in node feature space for enhanced performance.

In the meantime, adversarial data augmentation, which happens in the input feature space, is known to boost neural network robustness and promote resistance to adversarially chosen inputs (Goodfellow et al., 2014; Madry et al., 2017). Despite the wide belief that adversarial training harms standard

generalization and leads to worse accuracy (Tsipras et al., 2018; Balaji et al., 2019), recently a growing amount of attention has been paid to using adversarial perturbations to augment datasets and ultimately alleviate overfitting. For example, Volpi et al. (2018) showed adversarial data augmentation is a data-dependent regularization that could help generalize to out-of-distribution samples, and its effectiveness has been verified in domains including computer vision (Xie et al., 2020), language understanding (Zhu et al., 2019; Jiang et al., 2019), and visual question answering (Gan et al., 2020). Despite the rich literature about adversarial training of GNNs for security purposes (Zügner et al., 2018; Dai et al., 2018; Bojchevski & Günnemann, 2019; Zhang & Zitnik, 2020), it remains unclear how to effectively and efficiently improve GNNs' clean accuracy using adversarial augmentation.

Present work. We propose **FLAG**, **F**ree **L**arge-scale **A**dversarial Augmentation on **G**raphs, to tackle the overfitting problem. While existing literature focuses on modifying graph structures to augment datasets, FLAG works purely in the node feature space by adding gradient-based adversarial perturbations to the input node features with graph structures unchanged. FLAG leverages "free" methods (Shafahi et al., 2019) to conduct efficient adversarial training so that it is highly scalable to large-scale datasets. We verify the effectiveness of FLAG on the *Open Graph Benchmark* (OGB) (Hu et al., 2020), which is a collection of large-scale, realistic, and diverse graph datasets for both node and graph property prediction tasks. We conduct extensive experiments across OGB datasets by applying FLAG to prestigious GNN models, which are GCN, GraphSAGE, GAT, and GIN (Kipf & Welling, 2016; Hamilton et al., 2017; Veličković et al., 2017; Xu et al., 2019) and show that FLAG brings consistent and significant improvements. For example, FLAG lifts the test accuracy of GAT on `ogbn-products` by an absolute value of 2.31%. DeeperGCN (Li et al., 2020) is another strong baseline that achieves top performance on several OGB benchmarks. FLAG enables DeeperGCN to generalize further and reach new state-of-the-art performance on `ogbg-molpcba` and `ogbg-ppa`. FLAG is simple (adding just a dozen lines of code), general (can be directly applied to any GNN model), versatile (works in both transductive and inductive settings), and efficient (able to bring salient improvement at tractable or even no extra cost). Our main contributions are summarized as follows:

- We propose adversarial perturbations as a data augmentation in the input node feature space to efficiently boost GNNs' performance. The resulting FLAG framework is a scalable and flexible augmentation scheme for GNNs, which is easy to implement and applicable to any GNN architecture for both node and graph classification tasks.
- We advance the state-of-the-art on a number of large-scale OGB datasets, often by large margins.
- We provide a detailed analysis and deep insights on the effects adversarial augmentation has on GNNs.

2 PRELIMINARIES

Graph Neural Networks (GNNs). We denote a graph as $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with initial node features $\boldsymbol{x}_v$ for $v \in \mathcal{V}$ and edge features $\boldsymbol{e}_{uv}$ for $(u, v) \in \mathcal{E}$. GNNs are built on graph structures to learn representation vectors $\boldsymbol{h}_v$ for every node $v \in \mathcal{V}$ and a vector $\boldsymbol{h}_{\mathcal{G}}$ for the entire graph $\mathcal{G}$. The k-th iteration of message passing, or the k-th layer of GNN forward computation is:

$$\boldsymbol{h}_v^{(k)} = \text{COMBINE}^{(k)}\left(\boldsymbol{h}_v^{(k-1)},\ \text{AGGREGATE}^{(k)}\left(\left\{\left(\boldsymbol{h}_v^{(k-1)}, \boldsymbol{h}_u^{(k-1)}, \boldsymbol{e}_{uv}\right) : u \in \mathcal{N}(v)\right\}\right)\right), \quad (1)$$

where $\boldsymbol{h}_v^{(k)}$ is the embedding of node v at the k-th layer, $\boldsymbol{e}_{uv}$ is the feature vector of the edge between node u and v, $\mathcal{N}(v)$ is node v's neighbor set, and $\boldsymbol{h}_v^{(0)} = \boldsymbol{x}_v$. COMBINE$(\cdot)$ and AGGREGATE$(\cdot)$ are functions parameterized by neural networks. To simplify, we view the holistic message passing pipeline as an end-to-end function $f_{\boldsymbol{\theta}}(\cdot)$ built on graph $\mathcal{G}$:

$$\boldsymbol{H}^{(K)} = f_{\boldsymbol{\theta}}(\boldsymbol{X}; \mathcal{G}), \quad (2)$$

where $\boldsymbol{X}$ is the input node feature matrix. After K rounds of message passing we get the final-layer node matrix $\boldsymbol{H}^{(K)}$. To obtain the representation of the entire graph $\boldsymbol{h}_{\mathcal{G}}$, the permutation-invariant READOUT$(\cdot)$ function pools node features from the final iteration K as:

$$\boldsymbol{h}_{\mathcal{G}} = \text{READOUT}\left(\left\{\boldsymbol{h}_v^{(K)} \mid v \in \mathcal{V}\right\}\right), \quad (3)$$

Additionally from the spectral convolution point of view, the k-th layer of GCN is:

$$\boldsymbol{I} + \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}} \rightarrow \tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}, \boldsymbol{S} = \tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}, \tag{4}$$

$$\boldsymbol{H}^{(k+1)} = \sigma\left(\boldsymbol{S}\boldsymbol{H}^{(k)}\Theta^{(k)}\right), \tag{5}$$

where $\boldsymbol{H}^{(k)}$ is the node feature matrix of the k-th layer with $\boldsymbol{H}^0 = \boldsymbol{X}$, Θ^k is the trainable weight matrix of layer k, and σ is the activation function. $\boldsymbol{D}$ and $\boldsymbol{A}$ denote the diagonal degree matrix and adjacency matrix, respectively. Here, we view $\boldsymbol{S}$ as a normalized adjacency matrix with self-loops added.

Adversarial training. Standard adversarial training seeks to solve the min-max problem as:

$$\min_{\boldsymbol{\theta}} \ \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}}\left[\max_{\|\boldsymbol{\delta}\|_p\leq\epsilon} L\left(f_{\boldsymbol{\theta}}(\boldsymbol{x}+\boldsymbol{\delta}), y\right)\right], \tag{6}$$

where $\mathcal{D}$ is the data distribution, y is the label, $\|\cdot\|_p$ is some ℓ_p-norm distance metric, ϵ is the perturbation budget, and L is the objective function. Madry et al. (2017) showed that this saddle-point optimization problem could be reliably tackled by Stochastic Gradient Descent (SGD) for the outer minimization and Projected Gradient Descent (PGD) for the inner maximization. In practice, the typical approximation of the inner maximization under an l_∞-norm constraint is as follows,

$$\boldsymbol{\delta}_{t+1} = \Pi_{\|\boldsymbol{\delta}\|_\infty\leq\epsilon}\left(\boldsymbol{\delta}_t + \alpha\cdot\operatorname{sign}\left(\nabla_{\boldsymbol{\delta}} L\left(f_{\boldsymbol{\theta}}(\boldsymbol{x}+\boldsymbol{\delta}_t), y\right)\right)\right), \tag{7}$$

where perturbation $\boldsymbol{\delta}$ is updated iteratively, and $\Pi_{\|\boldsymbol{\delta}\|_\infty\leq\epsilon}$ performs projection onto the ϵ-ball in the l_∞-norm. For maximum robustness, this iterative updating procedure usually loops M times, which makes PGD computationally expensive. While there are M forward and backward steps within the process, $\boldsymbol{\theta}$ gets updated just once using the final $\boldsymbol{\delta}_M$.

3 Proposed Method: FLAG

Adversarial training is a form of data augmentation. By hunting for and stamping out small perturbations that cause the classifier to fail, one may hope that adversarial training should be beneficial to standard accuracy (Goodfellow et al., 2014; Tsipras et al., 2018; Miyato et al., 2018). With an increasing amount of attention paid to leverage adversarial training for better clean performance in varied domains (Xie et al., 2020; Zhu et al., 2019; Gan et al., 2020), we conduct the first study on how to effectively generalize GNNs using adversarial data augmentation. Here we introduce FLAG, **F**ree **L**arge-scale **A**dversarial Augmentation on **G**raphs, to best exploit the power of adversarial augmentation. Note that our method differs from other augmentations for graphs in that it happens in the input node feature space.

Augmentation for "free". We leverage the "free" adversarial training method (Shafahi et al., 2019) to craft adversarial data augmentations. PGD is a strong but inefficient way to solve the inner maximization of (6). While computing the gradient for the perturbation $\boldsymbol{\delta}$, free training simultaneously computes the model parameter $\boldsymbol{\theta}$'s gradient. This "free" parameter gradient is then used to compute the ascent step. The authors proposed to train on the same minibatch M times in a row to simulate the inner maximization in (6), while compensating by performing M times fewer epochs of training. The resulting algorithm yields accuracy and robustness competitive with standard adversarial training, but with the same runtime as clean training.

Gradient accumulation. When doing "free" adversarial training, the inner/adversarial loop is usually run M times, each time computing both the gradient for $\boldsymbol{\delta}_t$ and $\boldsymbol{\theta}_{t-1}$. Rather than updating the model parameters in each loop, Zhang et al. (2019) proposed to accumulate the gradients for $\boldsymbol{\theta}_{t-1}$ during the inner loop and applied them all at once during the outer/parameter update. The same idea was used by Zhu et al. (2019), who proposed FreeLB to tackle this optimization issue on language understanding tasks. FreeLB ran multiple PGD steps to craft adversaries, and meanwhile accumulated the gradients $\nabla_{\boldsymbol{\theta}} L$ of model parameters. The gradient accumulation behavior can be approximated as optimizing the objective below:

$$\min_{\boldsymbol{\theta}} \ \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}}\left[\frac{1}{M}\sum_{t=0}^{M-1}\max_{\boldsymbol{\delta}_t\in\mathcal{I}_t} L\left(f_{\boldsymbol{\theta}}\left(\boldsymbol{x}+\boldsymbol{\delta}_t\right), y\right)\right], \tag{8}$$

Algorithm 1 FLAG: **F**ree **L**arge-scale **A**dversarial Augmentation on **G**raphs

Require: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$; input feature matrix $\boldsymbol{X}$; learning rate τ; ascent steps M; ascent step size α; training epochs N; forward function on graph $f_{\boldsymbol{\theta}}(\cdot)$ denoted in (2); $L(\cdot)$ as objective function. We omit the READOUT$(\cdot)$ function in (3) for the inductive scenario here.

Initialize $\boldsymbol{\theta}$
for epoch $= 1 \ldots N$ **do**
  $\boldsymbol{\delta}_0 \leftarrow U(-\alpha, \alpha)$ ▷ initialize from uniform distribution
  $\boldsymbol{g}_0 \leftarrow 0$
  for t $= 1 \ldots M$ **do**
    $\boldsymbol{g}_t \leftarrow \boldsymbol{g}_{t-1} + \frac{1}{M} \cdot \nabla_{\boldsymbol{\theta}} L\left(f_{\boldsymbol{\theta}}(\boldsymbol{X} + \boldsymbol{\delta}_{t-1}; \mathcal{G}), \boldsymbol{y}\right)$ ▷ $\boldsymbol{\theta}$ gradient accumulation
    $\boldsymbol{g}_{\boldsymbol{\delta}} \leftarrow \nabla_{\boldsymbol{\delta}} L\left(f_{\boldsymbol{\theta}}(\boldsymbol{X} + \boldsymbol{\delta}_{t-1}; \mathcal{G}), \boldsymbol{y}\right)$
    $\boldsymbol{\delta}_t \leftarrow \boldsymbol{\delta}_{t-1} + \alpha \cdot \boldsymbol{g}_{\boldsymbol{\delta}} / \left\|\boldsymbol{g}_{\boldsymbol{\delta}}\right\|_F$ ▷ perturbation $\boldsymbol{\delta}$ gradient ascent
  end for
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \tau \cdot \boldsymbol{g}_M$ ▷ model parameter $\boldsymbol{\theta}$ gradient descent
end for

where $\mathcal{I}_t = \mathcal{B}_{\boldsymbol{x}+\boldsymbol{\delta}_0}(\alpha t) \cap \mathcal{B}_{\boldsymbol{x}}(\epsilon)$. The gradient accumulation algorithm largely empowers FLAG to further improve GNN with efficient gradient usage for optimization.

Unbounded attack. Usually on images, the inner maximization is a constrained optimization problem. The largest perturbation one can add is bounded by the hyperparameter ϵ, typically 8/255 under the l_∞-norm. This ϵ encourages the visual imperceptibility of the perturbations, thus making defenses realistic and practical. However, graph node features or language word embeddings do not have such straightforward semantic meanings, which makes the selection of ϵ highly heuristic. In light of the positive effect of large perturbations on generalization (Volpi et al., 2018), and also to simplify hyperparameter search, FLAG drops the projection step when performing the inner maximization. Note that, although the perturbation is not bounded by an explicit ϵ, it is still implicitly bounded in the furthest distance that $\boldsymbol{\delta}$ can reach, i.e. the step size α times the number of ascending steps M.

Biased perturbation for node classification. Conventional conv nets treat each test sample independently during inference, whereas this is not the case in transductive graph learning scenarios. When classifying one target node, messages from the whole k-hop neighborhood are aggregated and combined into its embedding. It is natural to believe that a further neighbor should have lower impact, i.e. higher smoothness, on the final decision of the target node, which can also be intuitively reflected by the message passing view of GNNs in (1). To promote more invariance for further-away neighbors when doing node classification, we perturb unlabeled nodes with larger step sizes α_u than α_l for target nodes. We show the effectiveness of this biased perturbation in the ablation study section.

The overall augmentation pipeline is presented in Algorithm 1. Note that when doing transductive node classification, we use diverse step sizes α_l and α_u to craft adversarial augmentation for target and unlabeled nodes, respectively. In the following sections, we verify FLAG's effectiveness through extensive experiments. In addition, we provide detailed discussions for a deep understanding of the effects of adversarial augmentation.

4 EXPERIMENTS

In this section, we demonstrate FLAG's effectiveness through extensive experiments on the *Open Graph Benchmark* (OGB), which consists of a wide range of challenging large-scale datasets. Shchur et al. (2018); Errica et al. (2019); Dwivedi et al. (2020) showed that traditional graph datasets suffered from problems such as unrealistic and arbitrary data splits, highly limited data sizes, non-rigorous evaluation metrics, and common neglect of cross-validation, etc. In order to empirically study FLAG's effects in a fair and reliable manner, we conduct experiments on the newly released OGB (Hu et al., 2020) datasets, which have tackled those major issues and brought more realistic challenges to the graph research community. We refer readers to Hu et al. (2020) for detailed information on the OGB datasets.

Backbone	ogbn-products Test Acc	ogbn-proteins Test ROC-AUC	ogbn-arxiv Test Acc
GCN	-	**72.51±0.35**	71.74±0.29
+FLAG	-	71.71±0.50	**72.04±0.20**
GraphSAGE	78.70±0.36	**77.68 ±0.20**	71.49±0.27
+FLAG	**79.36±0.57**	76.57±0.75	**72.19±0.21**
GAT	79.45±0.59	-	73.65±0.11
+FLAG	**81.76±0.45**	-	**73.71±0.13**
DeeperGCN	80.98±0.20	85.80±0.17	71.92±0.16
+FLAG	**81.93±0.31**	**85.96±0.27**	**72.14±0.19**

Table 1: Node property prediction test performance on ogbn-products, ogbn-proteins, and ogbn-arxiv datasets. Blank denotes no statistics on the leaderboard.

Backbone	ogbn-products Test Acc
GAT	79.45±0.59
+FLAG$^{\diamond}$	**80.64±0.74**
+FLAG†	81.29±0.39
+FLAG‡	**81.76±0.45**

Backbone	ogbn-mag Test Acc
R-GCN	46.78±0.67
+FLAG	**47.37±0.48**

Table 2: Left: Test performance on ogbn-products with GAT as baseline. $\diamond$ denotes model trained in N/M epochs; † denotes $\alpha_u = \alpha_l$; ‡ denotes $\alpha_u = 2\alpha_l$. Right: Test performance on the heterogeneous OGB node property prediction dataset ogbn-mag.

Unless otherwise stated, all of the baseline test statistics come from the official OGB leaderboard website, and we conduct all of our experiments using publicly released implementations without touching the original model architecture or training setup. We report mean and std values from ten runs with different random seeds. Following common practice on this benchmark, we report the test performance associated with the best validation result. We choose the prestigious GCN, GraphSAGE, GAT, and GIN as our baseline models. In addition, we apply FLAG to the recent DeeperGCN model to demonstrate effectiveness. Our implementation always uses $M = 3$ ascent steps for simplicity. Following Goodfellow et al. (2014); Madry et al. (2017), we use sign$(\cdot)$ for gradient normalization. We leave exhaustive hyperparameter and normalization search for future research. All training hyperparameters and evaluation results can be found in the Appendix.

Node Property Prediction. We summarize the results of node classification in Table 1. On ogbn-products, GraphSAGE, GAT, and DeeperGCN all receive promising results with FLAG. We adopt neighbor sampling (Hamilton et al., 2017) as the mini-batch algorithm for GraphSAGE and GAT to make the experiments scalable. For DeeperGCN, we follow the original setup by Li et al. (2020) to randomly split the graph into clusters. Notably, FLAG yields a 2.31% test accuracy lift for GAT, making GAT competitive on the ogbn-products dataset. Because the graph size of ogbn-proteins is small, all models are trained in a full-batch manner. From Table 1 we can see that FLAG further enhances the performance of DeeperGCN but harms that of GCN and GraphSAGE. Considering the dataset's specialty of not having input node features, we provide detailed discussions on the effect of different node feature constructions later. We also do full-batch training on ogbn-arxiv, where FLAG enables GAT and DeeperGCN to reach 73.71% and 72.14% accuracy. Note that the GAT baseline is from the DGL (Wang et al., 2019) implementation, which differs from vanilla GAT with batch norm and label propagation incorporated. We reveal batch norm's influence in the discussion. ogbn-mag is a heterogeneous network where only "paper" nodes come with node features. We use the neighbor sampling mini-batch algorithm to train R-GCN and report its results in the right part of Table 2. Surprisingly, FLAG can also directly bring nontrivial accuracy improvement without special designs for heterogeneous graphs, which demonstrates its versatility.

Graph Property Prediction. Table 3 summarizes the test scores of GCN, GIN, and DeeperGCN on all four OGB graph property prediction datasets. "Virtual" means the model is augmented with virtual nodes (Li et al., 2017; Gilmer et al., 2017; Hu et al., 2020). As adversarial perturbations are crafted by gradient ascent, it would be unnatural to perturb discrete input node features. Following

Backbone	ogbg-molhiv Test ROC-AUC	ogbg-molpcba Test AP	ogbg-ppa Test Acc	ogbg-code Test F1
GCN	76.06±0.97	20.20±0.24	68.39±0.34	31.63±0.18
+FLAG	**76.83±1.02**	**21.16±0.17**	68.38±0.47	**32.09±0.19**
GCN-Virtual	**75.99±1.19**	24.24±0.34	68.57±0.61	32.63±0.13
+FLAG	75.45±1.58	**24.83±0.37**	**69.44±0.52**	**33.16*****±0.25**
GIN	75.58±1.40	22.66±0.28	68.92±1.00	31.63±0.20
+FLAG	**76.54±1.14**	**23.95±0.40**	**69.05±0.92**	**32.41±0.40**
GIN-Virtual	77.07±1.49	27.03±0.23	70.37±1.07	32.04±0.18
+FLAG	**77.48±0.96**	**28.34±0.38**	**72.45±1.14**	**32.96±0.36**
DeeperGCN	78.58±1.17	27.81$^{\natural}$±0.38	77.12±0.71	-
+FLAG	**79.42±1.20**	**28.42**$^{\star\natural}$**±0.43**	**77.52*****±0.69**	-

Table 3: Graph property test performance on ogbg-molhiv, ogbg-molpcba, ogbg-ppa, and ogbg-code datasets. $\star$ denotes state-of-the-art performance on the OGB leaderboard; $\natural$ denotes the existence of virtual nodes; blank denotes no statistics on the leaderboard.

Backbone	ogbn-products Test Acc
GraphSAGE w/ NS	78.70±0.36
+FLAG	**79.36±0.57**
GraphSAGE w/ Cluster	**78.97±0.33**
+FLAG	78.60±0.27
GraphSAGE w/ SAINT	79.08±0.24
+FLAG	**79.60±0.19**

Backbone	ogbn-products Test Acc
GAT	79.45±0.59
GAT+PGD	80.96±0.41
GAT+Free	79.42±0.84
GAT+FreeLB	81.28±0.73
GAT+FLAG	**81.76±0.45**

Table 4: Left: Test accuracy on ogbn-products with GraphSAGE trained with diverse mini-batch algorithms. Right: Test performance on ogbn-products with GAT trained with different adversarial augmentations.

Jin & Zhang (2019); Zhu et al. (2019), we firstly project discrete node features into the continuous space and then adversarially augment the hidden embeddings. On ogbg-molhiv, FLAG yields notable improvements, but when GCN has already been hurt by virtual nodes, FLAG appears to exaggerate the harm. Note that the test results on ogbg-molhiv all have relatively high variance compared with others, where randomness in the test result is more severe. On ogbg-molpcba, GIN-Virtual with FLAG receives an absolute value 1.31% test AP value increase, and DeeperGCN is further enhanced to retain its SOTA performance. On ogbg-ppa, FLAG further generalizes DeeperGCN and registers a new state-of-the-art test accuracy of 77.52%. On ogbg-code, FLAG boosts GCN-Virtual to a state-of-the-art test F1 score of 33.16. Besides node classification, FLAG's strong effects on graph classification prove its high versatility. In most cases, FLAG works well with virtual node augmentation to further enhance graph learning.

5 Ablation Studies and Discussions

Effects of biased perturbation. From the left part of Table 2, we see that there is a salient increase of accuracy when using a larger perturbation on unlabeled nodes, which verifies the effectiveness of biased perturbations.

Comparison with other adversarial training methods. The right part of Table 4 shows GAT's performance with different adversarial augmentations. For PGD and Free, we compute 8 ascent steps for the inner-maximization, while for FreeLB and FLAG we compute 3 steps. FLAG outperforms all other methods by a large margin.

Compatibility with mini-batch methods. Graph mini-batch algorithms are critical to training GNNs on large-scale datasets. We test how different algorithms will work with adversarial data augmentation with GraphSAGE as the backbone. From the left part of Table 4, we see that neighbor

Backbone	ogbn-arxiv Test Acc
GAT w/o BN	73.29±0.12
GAT w/ BN	73.65±0.11
GAT w/ BN +FLAG	**73.71±0.31**

Backbone	ogbn-products Test Acc
GAT w/o dropout	75.67±0.27
GAT w/ dropout	79.45±0.59
GAT w/ dropout +FLAG	**81.76±0.45**

Table 5: Left: Test Accuracy on the ogbn-arxiv dataset. Right: Test Accuracy on the ogbn-products dataset.

sampling (Hamilton et al., 2017) and GraphSAINT (Zeng et al., 2019) can all work with FLAG to further boost performance, while Cluster (Chiang et al., 2019) suffers an accuracy drop.

Compatibility with batch norm. The left part of Table 5 shows that batch norm works to generalize GAT, and FLAG works to push the improvement further. In the computer vision domain, Xie et al. (2020) proposed a new batch norm method that makes adversarial training further generalize large-scale CNN models. As there is growing attention on using batch norm on GNNs, it will also be interesting to see how to synergize adversarial augmentation with batch norm in future architectures.

Compatibility with dropout. Dropout is widely used in GNNs. The right part of Table 5 shows that, when trained without dropout, GAT accuracy drops steeply by a large margin. What's more, FLAG can further generalize GNN models together with dropout, similar to the phenomenon of image augmentations.

Towards going "free". FLAG introduces tractable extra training overhead. We empirically show that, when we decrease the total training epochs to make it as fast as the standard GNN training pipeline, FLAG still brings significant performance gains. The left part of Table 2 shows that FLAG with fewer epochs still generalizes the baseline. Empirically, on a single Nvidia RTX 2080Ti, 100-epoch vanilla GAT takes 88 mins, while FLAG$^\diamond$ in Table 2 takes 91 mins. We note that heuristics like early stopping and cyclic learning rates can further accelerate the adversarial training process (Wong et al., 2020), so there are abundant opportunities for further research on adversarial augmentation at lower or even no cost.

Towards going deep. Over-smoothing stops GNNs from going deep. FLAG shows its ability to boost both shallow and deep baselines, e.g. GCN and DeeperGCN. In the left part of Figure 1, we show FLAG's effects on generalization when a GNN goes progressively deeper. The experiments are conducted on ogbn-arxiv with GraphSAGE as the backbone, where a consistent improvement is evident.

What if there's no node feature? One natural question can be raised: what if no input node features are provided? ogbn-proteins is a dataset without input node features. Hu et al. (2020) proposed to average incoming edge features to obtain initial node features, while Li et al. (2020) used summation and achieved competitive results. Note that the GCN and GraphSAGE baselines in Table 1 use the "mean" node features as input and suffer an accuracy drop with FLAG; DeeperGCN leverages the "sum" and gets further improved. Interestingly, when DeeperGCN is trained with "mean" node features, it receives high invariance, so that even large magnitude perturbations will not change its result. The diverse behavior of adversarial augmentation implies the importance of node feature construction method selection.

6 Where Does the Boost Come from?

It is now widely believed that model robustness appears to be at odds with clean accuracy. Despite the proliferation of literature in using adversarial data augmentation to promote standard performance, it is still unsettled where the boost or detriment of adversarial training comes from.

Data distribution is the key. We conjecture that the diverse effects of adversarial training in different domains stem from differences in the input data distribution rather than model architectures. To ground our claim, we utilize FLAG to augment MLPs (an architecture where adversarial training has adverse effects in the image domain) on ogbn-arxiv, and successfully boost generalization. FLAG directly improves the test accuracy from $55.50 \pm 0.23\%$ to $56.02 \pm 0.19\%$. In general, adversarial training hurts the clean accuracy in image classification, but Tsipras et al. (2018) showed that

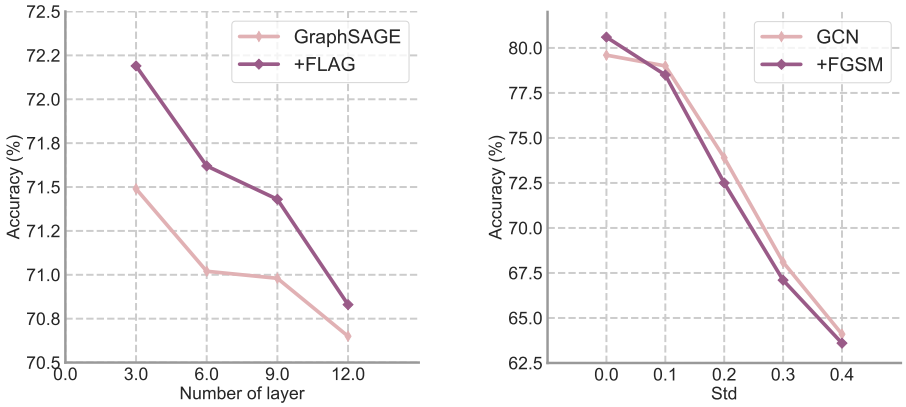

Figure 1: Left: Test accuracy on `ogbn-arxiv`. Right: Test accuracy on the `Cora` dataset.

CNNs could benefit from adversarial augmentations on MNIST, where the pixel values are closer to discrete distribution than other more natural image datasets. All these observations are consistent with our conjecture that data distribution has more to do with the effect of adversarial augmentation. Like one-hot word embeddings for language models, input node features usually come from discrete spaces, e.g., the bag-of-words binary features in `ogbn-products`. We believe that adversarial augmentation on discrete vs. continuous input features will lead to different effects. To illustrate, we provide a simple example on the `Cora` (Getoor, 2005) dataset. We choose FGSM to craft adversarial augmentation for a GCN. By adding Gaussian noise with standard deviation σ, we simulate node features drawn from a continuous distribution. The result is summarized in the right part of Figure 1. When $\sigma = 0$, the discrete distribution of node features persists. At this moment, a GCN with adversarial augmentation outperforms the non-augmented model. With increased noise level σ, the features are continuously distributed with large support and FGSM starts to harm the clean accuracy, which validates our conjecture.

7 Related Work

Existing graph regularizers mainly focus on augmenting graph structures by modifying edges (Rong et al., 2019; Hamilton et al., 2017; Chen et al., 2018). We propose to effectively augment graph data using adversarial perturbations. On large-scale image classification tasks, Xie et al. (2020) leveraged adversarial perturbations, along with new batch norm methods, to augment data. Zhu et al. (2019); Jiang et al. (2019) added adversarial perturbations in the embedding space and generalized language models further in the fine-tuning phase. Gan et al. (2020) showed that VQA model accuracy was further improved by adversarial augmentation. To clarify, FLAG is intrinsically different from the previous graph adversarial training methods (Feng et al., 2019; Deng et al., 2019; Jin & Zhang, 2019). Feng et al. (2019) proposed to reinforce local smoothness to make embeddings within communities similar. All three methods assigned pseudo-labels to test nodes during training time and utilized virtual adversarial training (Miyato et al., 2018) to make test node predictions similar to their pseudo-labels. This makes them workable for semi-supervised settings, but not for inductive tasks. Besides the original classification loss term, they all introduced KL loss into the final objective functions, which would at least double the GPU memory usage and make training less efficient and less scalable. In contrast, FLAG requires minimal extra space overhead and can directly work in the original training setup.

8 Conclusion

We propose FLAG (Free Large-scale Adversarial Augmentation on Graphs), a simple, scalable, and general data augmentation method for better GNN generalization. Like widely-used image augmentations, FLAG can be easily incorporated into any GNN training pipeline. FLAG yields consistent improvement over a range of GNN baselines, and reaches state-of-the-art performance

on the large-scale `ogbg-molpcba`, `ogbg-ppa`, and `ogbg-code` datasets. Besides extensive experiments, we also provide conceptual analysis to validate adversarial augmentation's different behavior on varied data types. The effects of adversarial augmentation on generalization are still not entirely understood, and we think this is a fertile space for future exploration.

REFERENCES

Yogesh Balaji, Tom Goldstein, and Judy Hoffman. Instance adaptive adversarial training: Improved accuracy tradeoffs in neural nets. *arXiv preprint arXiv:1910.08051*, 2019.

Aleksandar Bojchevski and Stephan Günnemann. Adversarial attacks on node embeddings via graph poisoning. In *International Conference on Machine Learning*, pp. 695–704. PMLR, 2019.

Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018.

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.

Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. Adversarial attack on graph structured data. *arXiv preprint arXiv:1806.02371*, 2018.

Zhijie Deng, Yinpeng Dong, and Jun Zhu. Batch virtual adversarial training for graph convolutional networks. *arXiv preprint arXiv:1902.09192*, 2019.

Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. *arXiv preprint arXiv:1912.09893*, 2019.

Fuli Feng, Xiangnan He, Jie Tang, and Tat-Seng Chua. Graph adversarial training: Dynamically regularizing based on graph structure. *IEEE Transactions on Knowledge and Data Engineering*, 2019.

Zhe Gan, Yen-Chun Chen, Linjie Li, Chen Zhu, Yu Cheng, and Jingjing Liu. Large-scale adversarial training for vision-and-language representation learning. *arXiv preprint arXiv:2006.06195*, 2020.

Victor Garcia and Joan Bruna. Few-shot learning with graph neural networks. *arXiv preprint arXiv:1711.04043*, 2017.

Lise Getoor. Link-based classification. In *Advanced methods for knowledge discovery from complex data*, pp. 189–207. Springer, 2005.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pp. 1024–1034, 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. *arXiv preprint arXiv:2005.00687*, 2020.

Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. *arXiv preprint arXiv:1911.03437*, 2019.

Hongwei Jin and Xinhua Zhang. Latent adversarial training of graph convolution networks. In *ICML Workshop on Learning and Reasoning with Graph-Structured Representations*, 2019.

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

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Chang Li and Dan Goldwasser. Encoding social information with graph convolutional networks forpolitical perspective detection in news media. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 2594–2604, 2019.

Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. Deepergcn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739*, 2020.

Junying Li, Deng Cai, and Xiaofei He. Learning graph-level representation for drug discovery. *arXiv preprint arXiv:1709.03741*, 2017.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):1979–1993, 2018.

Jiezhong Qiu, Jian Tang, Hao Ma, Yuxiao Dong, Kuansan Wang, and Jie Tang. Deepinf: Social influence prediction with deep learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2110–2119, 2018.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2019.

Ali Shafahi, Mahyar Najibi, Mohammad Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! In *Advances in Neural Information Processing Systems*, pp. 3358–3369, 2019.

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

Yantao Shen, Hongsheng Li, Shuai Yi, Dapeng Chen, and Xiaogang Wang. Person re-identification with deep similarity-guided graph neural network. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 486–504, 2018.

Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. *arXiv preprint arXiv:1805.12152*, 2018.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Riccardo Volpi, Hongseok Namkoong, Ozan Sener, John C Duchi, Vittorio Murino, and Silvio Savarese. Generalizing to unseen domains via adversarial data augmentation. In *Advances in neural information processing systems*, pp. 5334–5344, 2018.

Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

Jason Wei and Kai Zou. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. *arXiv preprint arXiv:1901.11196*, 2019.

Eric Wong, Leslie Rice, and J Zico Kolter. Fast is better than free: Revisiting adversarial training. *arXiv preprint arXiv:2001.03994*, 2020.

Cihang Xie, Mingxing Tan, Boqing Gong, Jiang Wang, Alan L Yuille, and Quoc V Le. Adversarial examples improve image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 819–828, 2020.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=ryGs6iA5Km`.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 974–983, 2018.

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

Dinghuai Zhang, Tianyuan Zhang, Yiping Lu, Zhanxing Zhu, and Bin Dong. You only propagate once: Accelerating adversarial training via maximal principle. *arXiv preprint arXiv:1905.00877*, 2019.

Xiang Zhang and Marinka Zitnik. Gnnguard: Defending graph neural networks against adversarial attacks. *arXiv preprint arXiv:2006.08149*, 2020.

Long Zhao, Xi Peng, Yu Tian, Mubbasir Kapadia, and Dimitris N Metaxas. Semantic graph convolutional networks for 3d human pose regression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3425–3435, 2019.

Chen Zhu, Yu Cheng, Zhe Gan, Siqi Sun, Thomas Goldstein, and Jingjing Liu. Freelb: Enhanced adversarial training for language understanding. *arXiv preprint arXiv:1909.11764*, 2019.

Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2847–2856, 2018.

Appendix

A FLAG PyTorch Implementation

```
#M as ascent steps, alpha as ascent step size
#X denotes input node features, y denotes labels
def flag(model, X, y, optimizer, criterion, M, alpha) :
    model.train()
    optimizer.zero_grad()

    pert = torch.FloatTensor(*X.shape).uniform_(-alpha, alpha)
    pert.requires_grad_()
    out = model(X+pert)
    loss = criterion(out, y)/M

    for _ in range(M-1):
        loss.backward()
        pert_data = pert.detach() + alpha*torch.sign(pert.grad.detach())
        pert.data = pert_data.data
        pert.grad[:] = 0
        out = model(X+pert)
        loss = criterion(out, y)/M

    loss.backward()
    optimizer.step()
```

B FULL STATISTICS

Here we summarize our main experiment results on both node and graph classification tasks. Hyperparameters for crafting adversarial augmentations are listed in the table. For other training setups of backbones, we refer readers to the public website of the OGB leaderboard.

Table 6: `ogbn-products`

Backbone	Test Acc	Val Acc	α_l	α_u/α_l	M
GraphSAGE	78.70±0.36	91.70±0.09	-	-	-
+FLAG	**79.36±0.57**	92.05±0.07	8e-03	2	3
GAT	79.45±0.59	-	-	-	-
+FLAG	**81.76±0.45**	92.51±0.06	5e-03	2	3
DeeperGCN	80.98±0.20	92.38±0.09	-	-	-
+FLAG	**81.93±0.31**	92.21±0.37	5e-03	2	3

Table 7: `ogbn-proteins`

Backbone	Test ROC-AUC	Val ROC-AUC	α_l	α_u/α_l	M
GCN	**72.51±0.35**	79.21±0.18	-	-	-
+FLAG	71.71±0.50	78.93±0.16	1e-03	1	3
GraphSAGE	**77.68 ±0.20**	83.34±0.13	-	-	-
+FLAG	76.57±0.75	82.84±0.17	1e-03	1	3
DeeperGCN	85.80±0.17	71.92±0.16	-	-	-
+FLAG	**85.96±0.27**	91.32±0.22	8e-03	1	3

Table 8: `ogbn-arxiv`

Backbone	Test Acc	Val Acc	α_l	α_u/α_l	M
MLP	55.50±0.23	57.65±0.17	-	-	-
+FLAG	**56.02±0.19**	58.17±0.11	2e-03	1	3
GCN	71.74±0.29	73.00±0.17	-	-	-
+FLAG	**72.04±0.20**	73.30±0.10	1e-03	1	3
GraphSAGE	71.49±0.27	72.77±0.16	-	-	-
+FLAG	**72.19±0.21**	73.49±0.09	1e-03	1	3
GAT	73.65±0.11	75.04±0.06	-	-	-
+FLAG	**73.71±0.13**	74.96±0.10	1e-03	2	3
DeeperGCN	71.92±0.16	72.62±0.14	-	-	-
+FLAG	**72.14±0.19**	73.11±0.09	8e-03	1	3

Table 9: `ogbg-molhiv`

Backbone	Test ROC-AUC	Val ROC-AUC	α	M
GCN	76.06±0.97	82.04±1.41	-	-
+FLAG	**76.83±1.02**	81.76±0.87	1e-02	3
GCN-Virtual	**75.99±1.19**	83.84±0.91	-	-
+FLAG	75.45±1.58	83.83±1.15	1e-03	3
GIN	75.58±1.40	82.32±0.90	-	-
+FLAG	**76.54±1.14**	82.25±1.55	5e-03	3
GIN-Virtual	77.07±1.49	84.79±0.68	-	-
+FLAG	**77.48±0.96**	84.38±1.28	1e-03	3
DeeperGCN	78.58±1.17	84.27±0.63	-	-
+FLAG	**79.42±1.20**	84.25±0.61	1e-02	3

Table 10: `ogbg-molpcba`

Backbone	Test AP	Val AP	α	M
GCN	20.20±0.24	20.59±0.33	-	-
+FLAG	**21.16±0.17**	21.50±0.22	8e-03	3
GCN-Virtual	24.24±0.34	24.95 ±0.42	-	-
+FLAG	**24.83±0.37**	25.56±0.40	8e-03	3
GIN	22.66±0.28	23.05±0.27	-	-
+FLAG	**23.95±0.40**	24.51±0.42	8e-03	3
GIN-Virtual	27.03±0.23	27.98±0.25	-	-
+FLAG	**28.34±0.38**	29.12±0.26	8e-03	3
DeeperGCN	27.81±0.38	29.20±0.25	-	-
+FLAG	**28.42*±0.43**	29.52±0.29	8e-03	3

Table 11: `ogbg-ppa`

Backbone	Test Acc	Val Acc	α	M
GCN	68.39±0.34	64.97±0.34	-	-
+FLAG	68.38±0.47	64.98±0.45	2e-03	3
GCN-Virtual	68.57±0.61	65.11±0.48	-	-
+FLAG	**69.44±0.52**	66.38±0.55	5e-03	3
GIN	68.92±1.00	65.62±1.07	-	-
+FLAG	**69.05±0.92**	64.65±0.70	8e-03	3
GIN-Virtual	70.37±1.07	66.78±1.05	-	-
+FLAG	**72.45±1.14**	67.89±0.79	5e-03	3
DeeperGCN	77.12±0.71	73.13±0.78	-	-
+FLAG	**77.52*±0.69**	74.84±0.52	8e-03	3

Table 12: `ogbg-code`

Backbone	Test F1	Val F1	α	M
GCN	31.63±0.18	29.73±0.14	-	-
+FLAG	**32.09±0.19**	30.16±0.16	8e-03	3
GCN-Virtual	32.63±0.13	30.62±0.07	-	-
+FLAG	**33.16*±0.25**	30.99±0.16	8e-03	3
GIN	31.63±0.20	29.81±0.14	-	-
+FLAG	**32.41±0.40**	30.44±0.39	8e-03	3
GIN-Virtual	32.04±0.18	30.20±0.16	-	-
+FLAG	**32.96±0.36**	30.92±0.35	8e-03	3