

Diffusion-Guided Graph Data Augmentation

Supplementary Material

A How Proposed D-GDA is Different from the Existing SOTA Methods?

Table 15 presents an extended comparison of D-GDA against 30 State-Of-The-Art (SOTA) methods, evaluating key aspects such as basic building blocks, supported tasks, robustness assessments, learning settings, and test-time augmentation. It table is an extension of Table 1 in the main paper.

Table 15: (Table 1 extension) A comparison of SOTA GDA methods: 1) **Basic building blocks:** graph auto-encoder (GAE), Graph Variational auto-encoder (GVAE), and Diffusion models. 2) **Supported tasks:** Node Classification (NC), Link Prediction (LP), and Graph Classification (GC). 3) **Robustness evaluations:** adversarial robustness (Adv) and machine learning safety measures (MSM). 4) **Learning Settings:** semi-supervised, Supervised, and Long-tailed (LT). 5) **Test-Time Augmentation** (TTA).

Methods	Basic Building Blocks			Supported Tasks			Robustness		Learning Setting			TTA
	GAE	GVAE	Diff.	NC	LP	GC	Adv	MSM	Semi	Sup.	LT	
DropEdge [51]	×	×	×	✓	×	×	×	×	✓	×	×	×
AdaEdge [7]	×	×	×	✓	×	×	×	×	✓	×	×	×
NodeAug [75]	×	×	×	✓	×	×	×	×	✓	×	×	×
GAug [85]	✓	×	×	✓	×	×	×	×	✓	×	×	×
Graph Mixup [68]	×	×	×	✓	×	✓	×	×	✓	✓	×	×
ReNode [8]	×	×	×	✓	×	×	×	×	×	×	✓	×
GraphSMOTE [82]	×	×	×	✓	×	×	×	×	×	×	✓	×
SR+DR [57]	×	×	×	✓	×	×	×	×	✓	×	×	×
GraphENS [47]	×	×	×	✓	×	×	×	×	✓	×	✓	×
FLAG [30]	×	×	×	✓	✓	✓	×	×	✓	✓	×	×
LA [41]	×	✓	×	✓	✓	✓	×	×	✓	✓	×	×
NASA [3]	×	×	×	✓	×	×	×	×	✓	×	×	×
CFLP [84]	×	×	×	×	✓	×	×	×	×	✓	×	×
TAM(G-ENS) [56]	×	×	×	✓	×	×	×	×	×	×	✓	×
G-Mixup [17]	×	×	×	×	×	✓	×	×	×	×	✓	×
GraphSHA [36]	×	×	×	✓	×	×	×	×	×	×	✓	×
DropMessage [13]	×	×	×	✓	✓	×	×	×	✓	✓	×	×
S-Mixup [25]	×	×	×	✓	×	×	×	×	✓	×	×	×
DropEdge++ [16]	×	×	×	✓	×	×	×	×	✓	✓	×	×
GraphPatcher [24]	×	×	×	✓	×	×	×	×	✓	×	×	✓
iGraphMix [22]	×	×	×	✓	✓	×	×	×	✓	✓	×	×
SkipNode [42]	×	×	×	✓	×	×	×	×	✓	✓	×	×
GeoMix [86]	×	×	×	✓	×	×	×	×	✓	×	×	×
RGDA [38]	×	×	×	✓	×	✓	×	×	✓	✓	×	×
xAI-DropEdge [11]	×	×	×	✓	×	×	×	×	✓	×	×	×
DCT [39]	×	×	✓	×	×	✓	×	×	×	✓	×	×
LeDA [40]	×	×	×	✓	×	×	✓	×	✓	×	×	×
GM [58]	×	×	×	×	✓	×	×	×	×	✓	×	×
DoG [66]	✓	×	✓	✓	×	×	×	×	✓	×	×	×
GraphVCM [49]	×	×	×	✓	×	×	×	×	×	×	✓	×
D-GDA (ours)	×	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

B Details of Experiments Performed using Proposed D-GDA

B.1 Datasets' Details

Node Classification: We evaluate our proposed method on four small-scale and two large-scale datasets. Cora, Citeseer, and Pubmed [53] are citation networks where nodes are papers and edges represent citation links. Node features are bag-of-words representation of papers. Flickr [44] is a social network in which nodes are users and edges represent user interaction from an image and video hosting website. OGBN-Arxiv [21] is a directed graph, representing the citation network between all Computer Science (CS) arXiv papers. Each node is an arXiv paper and each directed edge indicates

that one paper cites another one. Each paper comes with a 128-dimensional feature vector obtained by averaging the embeddings of words in its title and abstract. OGBN-Products [21] is a co-purchasing network having products as nodes and the edge indicates that the products are bought together. Table 16 summarizes the dataset statistics.

Table 16: Summary statistics of node classification evaluation datasets

Datasets	#Nodes	#Edges	#Classes	Train/Valid/Test split
Cora	2,708	5,278	7	140/500/1,000 [28]
Citeseer	3,327	4,552	6	120/500/1,000 [28]
Flickr	7,575	2,39,738	9	757/1,515/5,303 [85]
Pubmed	19,717	44,338	3	60/500/1,000 [28]
OGBN-Arxiv	169,343	1,166,243	40	90,941/29,799/48,603 [21]
OGBN-Products	2,449,029	61,859,140	47	196,615/39,323/2,213,091 [21]

Link Prediction: We evaluate our proposed method on five datasets: ogbl-collab [65] ogbl-ddi [70], Cora, Citeseer, and Pubmed [53]. Ogbl-collab [65] is an undirected collaboration network between authors indexed by MAG. Each node represents an author, and edges indicate collaborations between authors. All nodes have 128-dimensional features, obtained by averaging the word embeddings of papers published by the authors. Each edge is associated with two pieces of meta-information: the year and the edge weight, representing the number of co-authored papers published in that year. The task is to predict future author collaborations given past collaborations. Ogbl-ddi [70] is an undirected graph representing the drug-drug interaction network. Each node represents an FDA-approved or experimental drug, and edges represent interactions between drugs, indicating that the joint effect of taking the two drugs together is significantly different from their independent effects. The task is to predict drug-drug interactions given information on known interactions. Cora, Citeseer, and Pubmed are citation networks. Table 17 summarizes the dataset details.

Table 17: Summary Statistics of link prediction evaluation datasets

Datasets	#Nodes	#Edges	#Features	Train/Val/Test
Ogbl-collab	235,868	1,285,465	128	92/04/04 [21]
Ogbl-ddi	4,267	1,334,889	-	80/10/10 [21]
Cora	2,708	5,278	1,433	85/05/10 [34]
Citeseer	3,327	4,552	3,703	85/05/10 [34]
Pubmed	19,717	44,338	500	85/05/10 [34]

Graph Classification: We evaluate our proposed method on four molecular property prediction datasets: ogbg-molSIDER, ogbg-ClinTox, ogbg-molHIV, and ogbg-molBACE [73]. Each graph represents a molecule, where nodes are atoms, and edges are chemical bonds. The input node features are 9-dimensional, including atomic number, chirality, and additional atom features such as formal charge and ring membership. Table 18 summarizes the key statistics for all five datasets.

Table 18: Summary Statistics of graph classification evaluation datasets

Datasets	#Graphs	Average #Nodes	Average #Edges	#Classes	Train/Val/Test
Ogbg-molSIDER	1,427	33.6	70.7	27	80/10/10 [21]
Ogbg-molClinTox	1,477	26.2	55.8	2	80/10/10 [21]
Ogbg-molHIV	41,127	25.5	27.5	2	80/10/10 [21]
Ogbg-molBACE	1,513	34.1	73.7	2	80/10/10 [21]

Datasets for Class Imbalance Evaluations: We evaluate our method on the Cora and Citeseer datasets [53] in a long-tailed setting. To assess models under conditions of high class imbalance, we create long-tailed citation networks following the methodology outlined in [10]. This involves adjusting the class distribution to follow a long-tailed pattern by systematically removing nodes, thereby increasing the imbalance ratio, which is defined as the ratio between the most frequent class and the least frequent class. To achieve this, we sort the classes in descending order by size and iteratively remove nodes from each class, starting with the major class. During the node removal process, we prioritize eliminating nodes with low degrees and removing the corresponding edges while striving to maintain graph connectivity. Table 19 summarizes the dataset statistics.

Table 19: Summary Statistics of Graph Class Imbalance Evaluation Datasets

Datasets	#Nodes	#Edges	#Classes	#Training Nodes Per Class
Cora	2,708	5,278	7	[34, 7, 158, 341, 73, 15, 3]
Citeseer	3,327	4,552	6	[3, 23, 371, 147, 58, 9]

Table 20: (Table 2 extension) Node classification performance comparison of D-GDA variants and SOTA methods on small-scale datasets. Best and 2nd best performances are in **bold** and underline, respectively.

Method	Cora	Flickr	Citeseer	Pubmed	Mean
GCN [28]	81.60±0.70	61.20±0.40	71.60±0.40	78.80±0.60	73.30
+DropEdge [51]	82.00±0.80	61.40±0.70	71.80±0.20	77.30±0.32	73.13
+AdaEdge [7]	81.90±0.70	61.20±0.50	72.80±0.70	77.40±0.50	73.33
+SR+DR [57]	83.44±0.34	-	71.76±0.15	80.64±0.12	78.61
+NodeAug [75]	82.10±0.90	-	71.40±0.60	78.80±0.40	77.43
+GAug[85]	83.60±0.50	62.20±0.30	73.30±1.10	80.20±0.30	74.83
+Graph Mixup [68]	73.80±0.02	-	64.30±0.04	76.60±0.18	71.57
+FLAG [30]	75.20±0.40	62.90±0.20	62.70±0.60	78.50±0.01	69.83
+LA [41]	84.60±0.50	<u>64.24±0.30</u>	74.70±0.50	81.70±0.70	76.31
+NASA [3]	<u>85.10±0.30</u>	-	<u>75.50±0.40</u>	80.20±0.30	<u>80.30</u>
+DropMessage [13]	83.33±0.11	53.55±0.23	71.83±0.09	79.20±0.06	71.98
+S-Mixup [25]	84.78±0.15	-	74.39±0.10	79.70±0.17	79.62
+DropEdge++ [16]	83.10±0.12	63.18±0.14	72.70±0.06	80.00±0.42	74.75
+GraphPatcher [24]	84.17±0.54	-	71.65±0.05	81.13±0.68	78.98
+xAI-DropEdge [11]	83.60±0.50	-	74.00±0.40	79.50±0.40	79.03
+iGraphMix [22]	83.78±0.42	53.61±0.12	73.67±0.61	79.93±0.60	72.75
+SkipNode [42]	82.00±0.40	50.73±0.09	69.60±0.50	77.50±0.70	69.96
+GeoMix [86]	84.08±0.74	-	75.06±0.36	80.06±0.93	79.73
+RGDA [38]	84.33±0.41	-	73.02±0.36	82.08±0.50	79.81
+LeDA [40]	78.60±0.32	62.64±0.32	67.50±0.18	79.70±0.02	72.11
+DoG [66]	84.00±0.30	-	73.60±0.40	<u>82.80±0.30</u>	80.13
+D-GDA (ours)	89.10±0.42	87.30±0.60	81.50±0.15	88.20±0.18	86.53

B.2 More Details of Experiments for Node Classification

B.2.1 Experiment Setup and Implementation Details

For TSS, we train a baseline 2-layer Graph Convolutional Network (GCN) with a hidden dimension of 32, trained using the Adam optimizer with a learning rate of 0.001 for 500 epochs. Early stopping is applied with a patience of 20 epochs to prevent overfitting. Next, we train a Graph Variational Autoencoder (GVAE) to learn latent representation. The GVAE consists of a 2-layer GCN encoder to learn node representations and two 2-layer Multi-Layer Perceptrons (MLPs) for decoding edge and feature information, respectively. We set the latent dimension to 64. The GVAE is optimized using a composite loss function: $\mathcal{L}_{\text{GVAE}} = \mathcal{L}_{\text{edge}} + \lambda_1 \mathcal{L}_{\text{feat}} + \lambda_2 \mathcal{L}_{\text{KL}}$, where $\lambda_1 = 0.3$ weights the feature reconstruction loss and $\lambda_2 = 0.01$ controls the KL-divergence term for regularization as recommended by the original authors [29]. The GVAE is trained for 1000 epochs with the Adam optimizer, using a learning rate of 0.01 and a weight decay of 5×10^{-5} . Following [33, 45], we apply edge masking at a rate of 0.3 and feature masking at a rate of 0.5 to enhance robustness during training. Finally, we train a Latent Diffusion Model (LDM) with a timestep $T = 1000$ and a hidden dimension of 64. The LDM is optimized using the Adam optimizer with a learning rate of 1×10^{-4} for 1000 epochs, generating high-quality augmented samples for the node classification task.

B.2.2 Improvements over GCN backbone

Table 20 presents a comprehensive comparison of D-GDA against existing SOTA methods using the GCN backbone. Across all evaluated datasets, D-GDA consistently outperforms both the baseline GCN and the second-best SOTA method, demonstrating significant performance improvements.

B.2.3 Improvements over GAT backbone

We evaluate the performance of D-GDA on the Graph Attention Network (GAT) backbone [62] using three benchmark citation network datasets: Cora, Citeseer, and Pubmed. Table 21 presents a comparative analysis of D-GDA against existing SOTA methods under the GAT architecture. D-GDA achieves substantial performance gains over both the baseline GAT model and current SOTA approaches. Specifically, D-GDA outperforms the GAT baseline by 8.1%, 8.5%, and 9.5%, and exceeds the previous SOTA performance by 4.7%, 4.3%, and 9.1% on Cora, Citeseer, and Pubmed, respectively. Given that GAT is a self-attention-based model, where performance is known to be highly sensitive to both graph connectivity and node features, the observed improvements highlight the effectiveness and compatibility of the augmentation strategies introduced by D-GDA within attention-based architectures.

Table 21: (Table 4 extension) Node classification performance comparison of D-GDA using additional backbones including GAT [62] and GSAGE [15].

	Method	Cora	Citeseer	Pubmed
GAT	Vanilla	81.3	70.5	79.4
	+DropEdge [51]	81.9	71.0	79.6
	+AdaEdge [7]	82.0	71.1	76.6
	+SR+DR [57]	83.5	72.4	79.4
	+NodeAug [75]	84.1	70.8	78.3
	+GAug [85]	82.2	71.6	79.3
	+LA [41]	84.7	74.7	79.8
	+DropMessage [13]	82.2	71.5	78.1
	+xAI-DropEdge [11]	82.6	72.8	78.8
	+iGraphMix [22]	83.2	72.3	78.4
	+LeDA [40]	83.5	72.1	79.5
	+SkipNode [42]	81.6	68.4	77.6
	+D-GDA (ours)	89.4	79.0	88.9
GSAGE	Vanilla	81.3	70.6	76.8
	+DropEdge [51]	81.6	70.8	77.1
	+AdaEdge [7]	81.5	71.3	77.2
	+NodeAug [75]	82.2	70.2	78.1
	+GAug [85]	83.2	72.7	78.5
	+LeDA [40]	82.2	72.3	77.6
	+SkipNode [42]	81.5	68.5	77.4
	+RGDA [38]	83.4	72.6	82.1
	+D-GDA (ours)	88.7	80.2	89.5

B.2.4 Improvements over GSAGE backbone

We evaluate the performance of D-GDA using GSAGE (Graph Sample and aggregatE) [15] backbone using three benchmark datasets, including Cora, Citeseer, and Pubmed. In Table 21 we compare our results with existing SOTA methods on GSAGE backbone. D-GDA consistently improves over both baseline GSAGE and SOTA methods. Specifically, D-GDA outperforms the GSAGE baseline by 7.4%, 9.6%, and 7.4%, and exceeds the previous SOTA performance by 5.3%, 7.5%, and 7.4% on Cora, Citeseer, and Pubmed, respectively.

B.3 Implementation Details for Link Prediction

To enhance data augmentation for link prediction tasks, for TSS, we train 2-layer Graph Convolutional Network (GCN) with a hidden dimension of 32 to generate node embeddings, followed by a simple

Table 22: (Table 7 extension) Balanced accuracy (%) comparison under class imbalance ratio of $\rho = 100$.

Method	Cora-LT	Citeseer-LT
GCN (Vanilla)	59.42±0.74	44.64±0.42
+Reweight	78.42±0.10	63.61±0.22
+cRT	76.54±0.22	60.60±0.25
+PC Softmax	77.30±0.13	62.15±0.45
+CB Loss	77.97±0.19	61.47±0.51
+Focal Loss	78.43±0.19	59.66±0.38
+ReNode [8]	67.61±0.13	47.78±0.31
+Upsample	75.52±0.11	55.05±0.11
+DR-GCN	73.90±0.29	56.18±1.10
+GraphSMOTE [82]	66.29±0.43	44.40±0.29
+GraphENS [47]	70.31±0.24	55.42±0.35
+TAM (G-ENS) [56]	72.10±0.23	57.15±0.34
+GraphSHA [36]	74.62±0.29	59.04±0.41
+GraphVCM [49]	75.81±0.42	60.53±1.37
+D-GDA (ours)	80.34±0.51	64.95±0.18

dot product to predict links. It is trained using the Adam optimizer with a learning rate of 0.001 for 1000 epochs, incorporating early stopping with a patience of 20 epochs to prevent overfitting. For all link prediction datasets, we set augmentation budget in the Target sample selector (TSS) to 20% of total number of training links for all datasets. The GVAE and LDM for link prediction are trained with the same parameters as those used for node classification.

B.4 Implementation Details for Graph Classification

To enhance data augmentation for graph classification tasks, we begin by training a Target Sample Selector (TSS) to identify the most informative graphs for augmentation. The TSS employs a 2-layer GCN with a hidden dimension of 32 to compute node embeddings, which are subsequently aggregated into graph-level representations using a mean readout layer. The model is optimized using the Adam optimizer with a learning rate of 0.001 and trained for up to 1000 epochs, employing early stopping with a patience of 20 epochs to mitigate overfitting. For all graph classification datasets, we set augmentation budget in the Target sample selector (TSS) to 20% of total number of training graphs for all datasets. This targeted approach ensures augmentation focuses on samples with the highest potential impact. The GVAE and LDM used for augmentation are trained with the same hyperparameters as those employed in the node classification setting.

B.5 Extended Experiments for Class Imbalance Handling Methods

The TSS, GVAE, and LDM components follow the same training procedures as described in the node classification setup (Section B.2).

B.5.1 Extended comparison with SOTA Methods

Table 22 provides an extended comparison of D-GDA with existing SOTA methods for handling class imbalance in the node classification task. This table serves as an extension of Table 7 presented in the main paper.

B.5.2 Imbalance Ratio Analysis

The Imbalance Ratio (IR) is defined as the ratio of the sample size of the largest majority class to that of the smallest minority class [10], with higher IR values indicating more severe class imbalance. Many existing graph imbalance handling methods attempt to mitigate this by generating synthetic nodes for all classes to match the majority class [82, 36, 47], thereby balancing the training data. However, such approaches often introduce unnecessary complexity and may lead to overfitting. In contrast, D-GDA effectively reduces class imbalance without enforcing class-level symmetry or generating synthetic nodes across all classes. At the core of D-GDA is the Targeted Sample

Selector (TSS), which identifies the nodes, regardless of their class, that are most likely to benefit from augmentation. This class-agnostic selection enables a more natural and targeted correction of imbalance. As a result, D-GDA leads to a significant reduction in the imbalance ratio. For instance, the IR drops from 113.67 to 19.55 on Cora, and from 123.67 to 14.6 on Citeseer, as shown in Table 23.

Table 23: Comparison of the augmentation budget (AugBudget) for reducing class imbalance ratio (IR) using D-GDA and SOTA methods.

Datasets	Methods	#Training Nodes per Class	IR	AugBudget
Cora	Original Data	[34, 7, 158, 341, 73, 15, 3]	113.67	-
	SOTA (Table 22) Augmentation	[341, 341, 341, 341, 341, 341, 341]	1	278%
	D-GDA Augmented Data	[85, 25, 160, 391, 78, 39, 20]	19.55	26.4%
Citeseer	Original Data	[3, 23, 371, 147, 58, 9]	123.67	-
	SOTA (Table 22) Augmentation	[371, 371, 371, 371, 371, 371]	1	264%
	D-GDA Augmented Data	[30, 66, 438, 208, 66, 36]	14.6	38.13%

B.6 D-GDA Improvements in ML Safety Measures (More Details)

We evaluate D-GDA across three key machine learning (ML) safety measures, including calibration, corruption robustness, and consistency, using three widely adopted benchmark datasets: Cora, Citeseer, and Pubmed. Table 8 in the main paper summarizes the results. All models are trained on clean versions of the datasets, i.e., the original data for the GCN baseline, and augmented versions for D-GDA and other GDA baselines. Performance is then assessed across the aforementioned safety tasks. Below, we define each safety measure, describe its objective, and provide details on the corresponding evaluation metrics.

Calibration: The goal of the calibration task is to classify nodes with calibrated prediction probabilities, i.e. matching the empirical frequency of correctness. In other words, a model that predicts with 80% confidence should be correct approximately 80% of the time. To evaluate this, we use the Root

Mean Squared (RMS) Calibration Error [19] defined as: $\sqrt{\mathbb{E}_c[(\mathcal{P}(Y = \hat{Y}|C = c) - c)^2]}$, where C is the classifier confidence that its prediction \hat{Y} is correct. We compute the RMS calibration error using 15 bins to estimate the empirical accuracy at different confidence levels.

Corruption: The objective of this task is to evaluate model robustness by classifying corrupted test nodes, with performance measured using the mean classification error. We assess D-GDA under four types of feature corruption: Gaussian noise, shot noise, impulse noise, and feature shift. Following the protocol from [18], we apply five severity levels to the Gaussian (G), shot (S), and impulse (I) noise corruptions to simulate varying degrees of degradation. For the feature shift corruption, we randomly select 10% of the test nodes and replace their features with those of a randomly chosen one-hop neighbor. It is important to emphasize that only test node features are corrupted, and the model is never trained on corrupted data. Below, we include the Python code used to generate these corrupted node features.

```

1 def gaussian_noise(feature, severity=1):
2     c = [0.04, 0.06, .08, .09, .10][severity - 1]
3     return feature + np.random.normal(size=feature.shape, scale=c)
4
5 def shot_noise(feature, severity=1):
6     c = [500, 250, 100, 75, 50][severity - 1]
7     return np.random.poisson(feature * c)
8
9 def impulse_noise(feature, severity=1):
10    c = [.01, .02, .03, .05, .07][severity - 1]
11    return sk.util.random_noise(feature, mode='s&p', amount=c)
12
13 def shift(node, features, adj):

```



```

14     one_order_nei = adj[node].nonzero()[1]
15     neighbors = np.append(one_order_nei, node)
16     swap_with = random.choice(neighbors)
17     corr_feat = features.clone()
18     corr_feat[node], corr_feat[swap_with] = corr_feat[swap_with], corr_feat[node]
19     return corr_feat

```

Consistency: This task evaluates a model’s ability to consistently classify similar versions of the same test node. For each node, we create a sequence S containing the original node’s features and several perturbed versions with increasing noise levels. The goal is for the model to predict the same class for all nodes within a sequence S , even as the features become progressively noisier. We denote m perturbation sequences with $S = \{x_1^{(i)}, x_2^{(i)}, \dots, x_n^{(i)}\}$. Following [18], we set $n = 31$. To measure consistency, we use the mean flip rate (mFR), as defined in Eq. (2). The mFR corresponds to the probability that adjacent nodes within a sequence S of increasing noise levels have different predicted classes.

$$\text{mFR} = \frac{1}{m(n-1)} \sum_{i=1}^m \sum_{j=2}^n \mathbb{1}(f(x_j^{(i)}) \neq f(x_1^{(i)}), \quad (2)$$

where, $f(\cdot)$ is the trained classifier and x_1 is the original unperturbed features. Below is the Python code to obtain sequence S of original features.

```

1 def get_sequence(features, n=31):
2     seq = []
3     for i in range(1, n+1):
4         features = features + 0.02 * torch.randn_like(features)
5         seq.append(features)
6     return seq

```

B.7 D-GDA Adversarial Robustness Analysis (More Details)

To evaluate the robustness of D-GDA against adversarial attacks, we employ four well-established evasion attacks: Random (random edge flips), DICE [69] (removes intra-class edges and adds inter-class edges), GFAttack [6] (optimizes a low-rank loss to generate structural perturbations), and Meta-attack [88] (uses meta-gradient-based loss maximization). Each attack is applied at two perturbation levels, with perturbation ratios ($\sigma \in 0.05, 0.2$). All attacks are conducted under an evasion attack setting, where the graph structure is perturbed only at test time, leaving the training data unchanged. For GCN, the model is trained on clean data, whereas D-GDA is trained using its graph-augmented data. We leverage the GreatX: Graph Reliability Toolbox [32] to generate the adversarially perturbed graphs. Table 9 in the main paper summarizes the results.

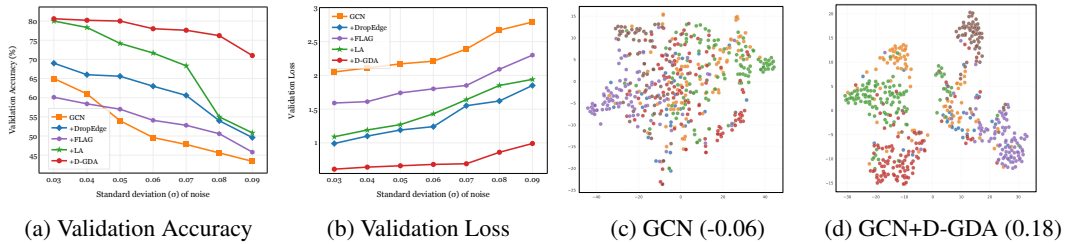


Figure 5: **Citeseer dataset:** (a) and (b): D-GDA promotes flatter minima in the loss landscape. (c) and (d): t-SNE visualizations of node embeddings, demonstrating improved class separability with D-GDA. Silhouette score is shown in parenthesis.

B.8 More insights to the improvements obtained by D-GDA

B.8.1 D-GDA promotes flatter minima for enhanced generalization

In addition to the Cora dataset results in the main paper, we evaluate D-GDA on the Citeseer and Pubmed datasets. Using a trained GCN, we compare D-GDA against DropEdge, FLAG, and LA under Gaussian noise perturbations of model parameters. As shown in Figure 5(a), D-GDA achieves higher accuracy and lower loss (Figure 5(b)) on Citeseer, indicating flatter minima that enhance generalization. t-SNE visualizations (Figure 5(c) and (d)) reveal more cohesive and well-separated clusters for D-GDA, with the silhouette score improving from -0.06 to 0.18. A similar trend is observed on Pubmed (Figure 6), where the silhouette score increases from 0.13 to 0.34, further confirming D-GDA’s ability to promote robust and generalizable representations.

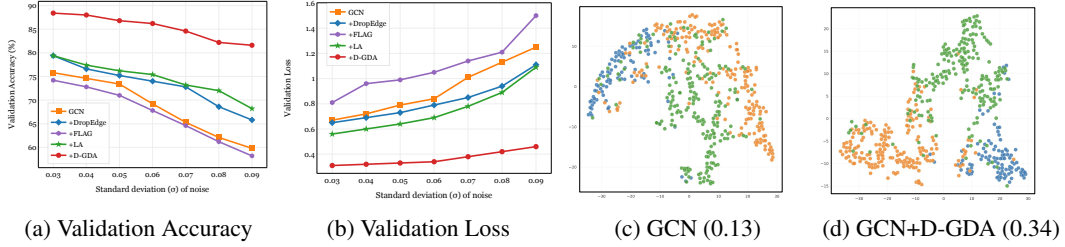


Figure 6: **Pubmed dataset:** (a) and (b): D-GDA promotes flatter minima in the loss landscape. (c) and (d): t-SNE visualizations of node embeddings, demonstrating improved class separability with D-GDA. Silhouette score is shown in parenthesis.

B.8.2 Oversmoothness analysis using MADGap measure (More Details)

To quantify the smoothness of graph representations, we employ the MADGap measure, as described by Chen et al. [7]. The results are presented in Table 10 of the main paper. Below, we outline the MADGap measure and its computation. The Mean Average Distance (MAD) assesses graph representation smoothness by computing the average distance between nodes. Given a graph representation matrix $H \in \mathbb{R}^{n \times d'}$ (where d' is the hidden dimension of the final GNN layer), we first calculate the distance matrix $D \in \mathbb{R}^{n \times n}$ using the cosine distance between all node pairs. We then filter target node pairs by element-wise multiplication with a mask matrix M^{tgt} to obtain $D^{\text{tgt}} = D \odot M^{\text{tgt}}$. The average distance for each row in D^{tgt} is computed as:

$$\bar{D}^{\text{tgt}} = \frac{\sum_{j=0}^n D_{ij}^{\text{tgt}}}{\sum_{j=0}^n \mathbb{1}(D_{ij}^{\text{tgt}})} \quad (3)$$

where $\mathbb{1}(\cdot)$ is the indicator function. The MAD score for the target node pairs is then calculated by averaging the non-zero \bar{D}_i^{tgt} values:

$$\text{MAD}^{\text{tgt}} = \frac{\sum_{j=0}^n \bar{D}_{ij}^{\text{tgt}}}{\sum_{j=0}^n \mathbb{1}(\bar{D}_{ij}^{\text{tgt}})} \quad (4)$$

Using the graph topology to approximate node categories, we compute the MADGap to evaluate oversmoothness:

$$\text{MADGap} = \text{MAD}^{\text{rnt}} - \text{MAD}^{\text{neb}}, \quad (5)$$

where MAD^{rnt} is the MAD for remote nodes (order ≥ 8) and MAD^{neb} is the MAD for neighboring nodes (order ≤ 3). A large positive MADGap indicates that nodes receive more useful information than noise, reflecting appropriate smoothing and good GNN performance. Conversely, a small or negative MADGap suggests oversmoothing, leading to degraded performance.

B.8.3 Test-Time Consistency and Diversity (More Details)

To evaluate the quality of augmented data, we adopt the consistency and diversity metrics proposed by Bo et al. [3] and extend them to the test set. We train two models, \mathcal{F}_θ and $\tilde{\mathcal{F}}_\theta : \mathbb{R}^d \rightarrow \mathbb{R}^c$, on

the original training data D_{train} and augmented data \tilde{D}_{train} , respectively, where d is the input feature dimension, c is the number of classes, and θ represents the learnable parameters. Both models are then used to predict on the test set D_{test} . Effective augmentations should enable $\tilde{\mathcal{F}}_{\theta}$ to achieve higher test accuracy and establish a distinct decision boundary compared to \mathcal{F}_{θ} .

Test-Time Consistency: We measure consistency as the accuracy of the augmented model on the test set, defined as $C_{\text{test}} = \text{Acc}(\tilde{\mathcal{F}}_{\theta}(D_{\text{test}}), Y_{\text{test}})$, where Y_{test} denotes the test data labels. A low C_{test} indicates that the augmentations are inconsistent with the original data, potentially reducing model accuracy. However, a high C_{test} does not necessarily imply high-quality augmentations, as it may reflect limited generalization.

Test-Time Diversity: To assess diversity, we compute the difference between the predictions of the original and augmented models using the Frobenius norm: $D_{\text{test}} = \|\tilde{\mathcal{F}}_{\theta}(D_{\text{test}}) - \mathcal{F}_{\theta}(D_{\text{test}})\|_F^2$. A low D_{test} suggests that the augmented data closely resembles the original data, offering little benefit to model generalization (Yin et al., 2019). Conversely, a high D_{test} does not guarantee correct augmentations, as it may introduce noise.

A robust augmentation strategy should balance high consistency (C_{test}) and sufficient diversity (D_{test}) to ensure both accuracy and improved generalization. Figure 7 illustrates the trade-off between diversity and consistency for D-GDA compared to existing SOTA augmentation methods. Across all datasets, D-GDA achieves a favorable balance between the two, suggesting that its performance gains may stem from this effective trade-off.

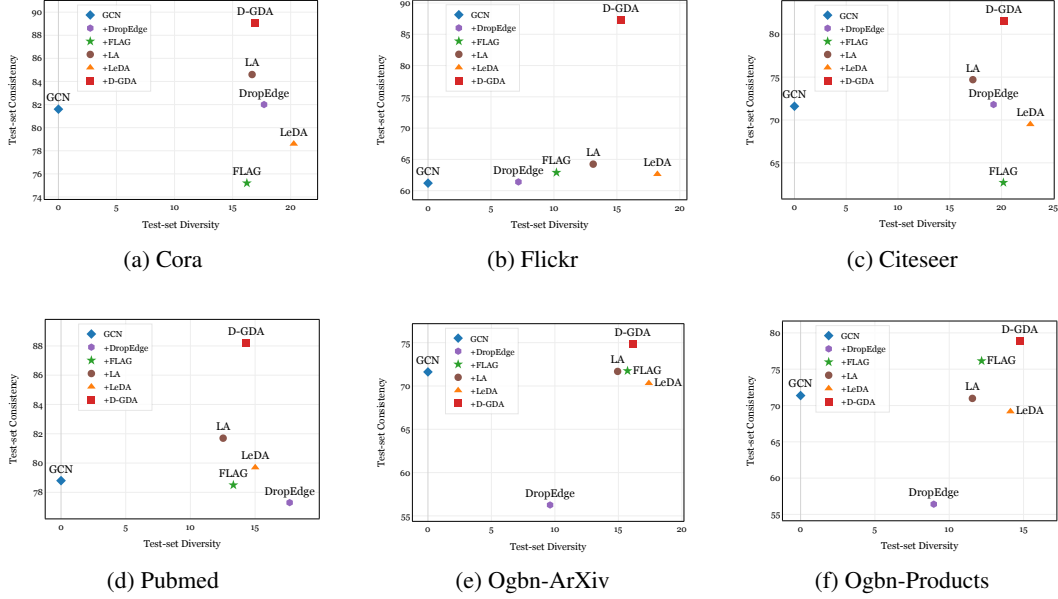


Figure 7: (Details of Figure 1b) Diversity vs. consistency comparison on six datasets. Methods falling in top-right corner shows a good balance of diversity vs. consistency

B.9 Additional Ablations

In this section, we present additional ablation studies beyond those in the main paper to provide a clearer understanding of various design choices.

B.9.1 Impact of Neighborhood Aggregation Method on Performance

To assess the impact of different neighborhood aggregation strategies, we experiment with two commonly used methods: mean and max aggregation. These strategies are used to compute the conditioning vector c_i^0 for augmentation generation. As shown in Table 24, the mean aggregation yields superior performance, indicating its effectiveness in capturing relevant neighborhood information for guiding the augmentation process.

Table 24: Impact of Neighborhood Aggregation Method on Performance

Aggregation Method	Cora	Citeseer	Pubmed
Max	87.5	78.8	84.7
Mean (ours)	89.1	81.5	88.2

B.9.2 Impact of Multi-Hop Neighborhood Condition on Performance

To assess the effect of multi-hop neighborhood condition on model performance, we conducted an ablation study by modifying the conditioning vector c_i^0 in the LDM to incorporate embeddings from 1-hop, 2-hop, and 3-hop neighboring nodes. The results, summarized in Table 25, show a decline in performance as neighborhood depth increases. This may be attributed to the changing class labels of nodes in deeper neighborhoods, which could prevent the synthetic node features from accurately capturing the target class’s characteristics. Notably, as depicted in Figure 8, increasing the neighborhood depth enhances test-time diversity but reduces test-time consistency.

Table 25: Impact of Multi-hop Neighborhood Aggregation on overall performance.

Conditioned On	Cora	Citeseer	Pubmed
1-hop neighbors (ours)	89.1	81.5	88.2
2-hop neighbors	88.2	79.9	87.9
3-hop neighbors	86.8	77.6	85.3

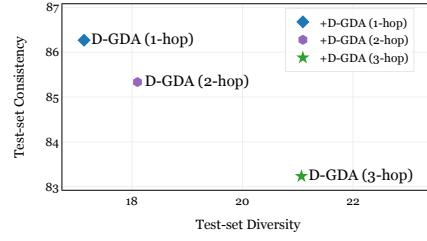


Figure 8: Multi-hop Neighborhood impact on test-time consistency and diversity

B.9.3 Hyperparameter Analysis

To evaluate the influence of hyperparameters on the overall performance of D-GDA, we conduct an ablation study focusing on the weighting parameters λ_1 and λ_2 in the GVAE loss function using Cora, Citeseer, and Pubmed datasets for the node classification task. The parameters λ_1 and λ_2 regulate the trade-off between feature reconstruction and latent space regularization, both of which are critical for generating high-quality augmented graph data. In this ablation, we systematically vary $\lambda_1 \in \{0.1, 0.3, 0.5\}$ and $\lambda_2 \in \{0.01, 0.05, 0.1\}$, while keeping all other components of D-GDA unchanged. For each pair of λ_1 and λ_2 values, we train the GVAE to generate augmented graphs, which are subsequently used to train the node classification model. Table 26 presents the results, reporting classification performance across different hyperparameter configurations. The analysis highlights the sensitivity of GVAE to the balance between feature reconstruction and regularization. For example, increasing λ_1 emphasizes feature reconstruction, which can improve node-level fidelity but may lead to overfitting. Conversely, higher λ_2 values impose stronger regularization on the latent space, promoting stability but potentially reducing the model’s capacity to capture nuanced structural information.

Table 26: Hyperparameter Analysis

λ_1	λ_2	Cora	Citeseer	Pubmed
0.1	0.01	88.38	79.92	86.45
	0.05	85.5	75.36	84.68
	0.1	82.28	72.83	81.73
0.3 (ours)	0.01 (ours)	89.1	81.5	88.2
	0.05	87.12	79.8	87.02
	0.1	82.52	73.86	84.7
0.5	0.01	88.56	80.13	87.14
	0.05	85.23	78.25	84.88
	0.1	82.48	74.6	81.92

B.10 Computational Complexity Comparisons of D-GDA with SOTA Non-Diffusion Methods

To study the scalability of D-GDA we compare the training and inference times of D-GDA with SOTA GDA methods for four datasets including Cora, Flickr, Pubmed, and Ogbn-Arxiv in Table 27. Training and inference time are reported using a 4x A16 GPU machine with 128GB RAM. Note that the execution times of SOTA Diffusion based GDA methods, could not be compared due to non-availability of their codes.

Table 27: Training and inference (Inf.) time (seconds) comparison of different SOTA methods with D-GDA. The number of nodes and node feature dimensionality varies across datasets. All algorithms are using early stopping however, the number of epochs used by different methods may vary during training.

Type	Method	Cora		Pubmed		Flickr		Ogbn-Arxiv	
		Train	Inf.	Train	Inf.	Train	Inf.	Train	Inf.
Non-Diffusion	GCN [28]	9.05	0.003	39.89	0.024	79.82	0.058	928.21	0.567
	DropEdge[51]	11.75	0.005	54.91	0.030	132.23	0.096	1786.40	0.813
	FLAG [30]	34.20	0.019	102.02	0.023	547.80	0.160	3307.50	0.815
Diffusion	D-GDA (ours)	37.72	0.019	229.42	0.024	328.17	0.160	2507.20	0.816

B.11 Limitations of the Proposed D-GDA Algorithm

A potential limitation of the proposed D-GDA framework is its training time computational cost and test-time memory overhead. It requires training GCN baseline, GVAE, and LDM for each dataset. This process is resource-intensive than some traditional augmentation methods, such as DropEdge. However, some traditional methods such as FLAG have also shown similar computational time. Being diffusion based framework, D-GDA requires GVAE and LDM during inference if test-time augmentation is employed. Without test-time augmentation its inference time remains almost the same as existing SOTA methods. Despite the higher computational overhead, the substantial performance improvements over SOTA methods justify the additional cost, making D-GDA a compelling choice for high-quality graph data augmentation.