# Graph Adversarial Diffusion Convolution

**Songtao Liu** [1] **Jinghui Chen** [1] **Tianfan Fu** [2] **Lu Lin** [1] **Marinka Zitnik** [3] **Dinghao Wu** [1]

## Abstract

This paper introduces a min-max optimization formulation for the Graph Signal Denoising (GSD) problem. In this formulation, we first maximize the second term of GSD by introducing perturbations to the graph structure based on Laplacian distance and then minimize the overall loss of the GSD. By solving the min-max optimization problem, we derive a new variant of the Graph Diffusion Convolution (GDC) architecture, called Graph Adversarial Diffusion Convolution (GADC). GADC differs from GDC by incorporating an additional term that enhances robustness against adversarial attacks on the graph structure and noise in node features. Moreover, GADC improves the performance of GDC on heterophilic graphs. Extensive experiments demonstrate the effectiveness of GADC across various datasets. Code is available at https://github.com/SongtaoLiu0823/GADC.

## 1. Introduction

Graph Neural Networks (GNNs) (Kipf & Welling, 2017; Hamilton et al., 2017; Veličković et al., 2018) have become a popular approach for graph-based tasks due to their powerful ability to learn node representations. They have demonstrated remarkable performance in various tasks, including traffic prediction (Guo et al., 2019), drug discovery (Dai et al., 2019), and recommendation system (Ying et al., 2018). The core principle of GNNs is the message-passing operation, which aggregates node features from neighboring nodes, thereby enhancing the smoothness of the learned node representations. As a result, GNN models produce predictions based on both the features of individual nodes and those of their immediate neighbors.

Graph Diffusion Convolution (GDC) (Gasteiger et al., 2019), a specialized GNN architecture, aggregates information from higher-order neighbors through generalized graph diffusion. Various GDC architectures (Li et al., 2019; Zhu & Koniusz, 2021; Liu et al., 2021b; Yang et al., 2021; Zhao et al., 2021; Jia & Benson, 2022) are derived from the Graph Signal Denoising (GSD) problem, formulated as:

$$\arg\min_{\mathbf{F}} \mathcal{L}(\mathbf{F}) := \|\mathbf{F}-\mathbf{X}\|_F^2 + \lambda \operatorname{tr}\left(\mathbf{F}^\top \tilde{\mathbf{L}}\mathbf{F}\right), \quad (1)$$

where $\mathbf{X} = \mathbf{X}^* + \mathbf{\Upsilon}$ is the observed noisy feature matrix, $\mathbf{\Upsilon} \in \mathbb{R}^{n\times d}$ is the noise matrix, $\mathbf{X}^*$ is the clean feature matrix, and $\tilde{\mathbf{L}} \in \mathbb{R}^{n\times n}$ is the normalized graph Laplacian matrix. A major strength of the GDC architecture is its ability to aggregate information from a larger set of neighbors, enhancing $\ell_2$-based graph smoothing (Liu et al., 2021b).

Despite significant advancements in GDC architectures, they rely heavily on the Laplacian matrix to derive the graph diffusion matrix. This dependency can be problematic when the graph structure is disrupted by adversarial attacks (Dai et al., 2018; Zügner et al., 2018; Zügner & Günnemann, 2019a; Jin et al., 2020). Such attacks can cause GDC to aggregate harmful information, disrupting node representations. Additionally, the limited number of neighbors restricts GDC's ability to effectively filter out large noise in node features within some graphs (Liu et al., 2021a). These challenges raise a crucial question: *Can we develop a versatile GDC architecture that addresses these issues effectively?* In this work, we provide a positive solution to this question by reformulating the GSD problem. Based on this reformulation, we design a new GDC architecture that mitigates the negative impact of adversarial attacks on the graph structure, noise in node features, and inconsistent edges on heterophilic graphs.

Inspired by the saddle point formulation of adversarial training (Madry et al., 2018), we propose a min-max variant of the GSD problem, called the Adversarial Graph Signal Denoising (AGSD) problem:

$$\arg\min_{\mathbf{F}} q(\mathbf{F}) := \left[\|\mathbf{F}-\mathbf{X}\|_F^2 + \lambda \cdot \max_{\mathbf{L}'} \operatorname{tr}\left(\mathbf{F}^\top \mathbf{L}'\mathbf{F}\right)\right], \quad (2)$$

where $\mathbf{L}'$ is the added perturbation by the adversary. In the inner optimization problem, we maximize the second term of the AGSD. Then, we minimize the loss function $q(\mathbf{F})$ in the outer optimization problem.

[1]The Pennsylvania State University [2]Rensselaer Polytechnic Institute [3]Harvard University. Correspondence to: Songtao Liu <skl5761@psu.edu>.

*Proceedings of the 41st International Conference on Machine Learning*, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

Unlike adversarial training, which requires projected gradient descent (PGD) to solve the inner maximization problem, our formulation has a closed-form solution for the inner optimization problem. Utilizing this closed-form solution, we solve the outer minimization problem to derive a new GDC architecture, Graph Adversarial Diffusion Convolution (GADC). This architecture introduces an additional term compared to GDC, resulting in more adaptive GDC variants that are resilient to adversarial perturbations in the edges and noise in node features. Moreover, our GADC enhances the performance of GDC on heterophilic graphs, where nodes from different classes are linked. Extensive experimental results across various datasets validate the effectiveness of our proposed algorithm. Additionally, our novel AGSD problem can inspire the development of advanced and reliable GNN architectures.

## 2. Preliminaries

**Notations.** Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ represent a undirected graph, where $\mathcal{V}$ is the set of vertices $\{v_1, \cdots, v_n\}$ with $|\mathcal{V}| = n$ and $\mathcal{E}$ is the set of edges. The adjacency matrix is defined as $\mathbf{A} \in \{0,1\}^{n\times n}$, and $\mathbf{A}_{i,j} = 1$ if and only if $(v_i, v_j) \in \mathcal{E}$. Let $\mathcal{N}_i = \{v_j | \mathbf{A}_{i,j} = 1\}$ denote the neighborhood of node $v_i$ and $\mathbf{D}$ denote the diagonal degree matrix, where $\mathbf{D}_{i,i} = \sum_{j=1}^{n} \mathbf{A}_{i,j}$. The normalized Laplacian matrix of a graph is defined as $\mathbf{L} = \text{Laplacian}(\mathbf{A}) = \mathbf{I}_n - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$, where $\mathbf{I}_n$ is an $n \times n$ identity matrix. The feature matrix is denoted as $\mathbf{X} \in \mathbb{R}^{n\times d}$. $\text{tr}(\cdot)$ denotes the trace of the matrix.

**Spectral Graph Convolution.** Spectral convolution involves multiplying a signal $\mathbf{x}$ by a Fourier domain filter $g_\phi$, parameterized by coefficients $\phi \in \mathbb{R}^n$. This is expressed as:

$$g_\phi(\mathbf{L}) \star \mathbf{x} = \mathbf{U} g^*_\phi(\mathbf{\Lambda}) \mathbf{U}^\top \mathbf{x}, \tag{3}$$

where $g_\phi$ can be approximated by a truncated expansion. A common approach is to use Chebyshev polynomials up to the $K$-th order, resulting in the following approximation:

$$g^*_\phi(\mathbf{\Lambda}) \approx \sum_{k=0}^{K} \phi_k T_k(\tilde{\mathbf{\Lambda}}). \tag{4}$$

**Graph Convolution Network (GCN).** GCNs approximate spectral graph convolution using first-order Chebyshev polynomials. By setting $K = 1$ and $\phi_0 = -\phi_1$, and approximating $\lambda_n \approx 2$, the convolution becomes $g_\phi(\mathbf{L}) \star \mathbf{x} = \left(\mathbf{I}_n + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\right)\mathbf{x}$. Introducing self-loops and renormalization trick modifies this to $\tilde{\boldsymbol{\mathcal{A}}} = \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$, where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_n$ and $\tilde{\mathbf{D}} = \mathbf{D} + \mathbf{I}_n$. The GCN layer is then defined as:

$$\mathbf{H}^{(l+1)} = \sigma\left(\tilde{\boldsymbol{\mathcal{A}}}\mathbf{H}^{(l)}\mathbf{\Theta}^{(l)}\right). \tag{5}$$

Here, $\sigma$ is the activation function and $\mathbf{\Theta}^{(l)}$ are the trainable parameters in the $l$-th layer.

**Adversarial Training.** Madry et al. (2018) study the adversarial robustness of neural networks from the perspective of robust optimization as follows:

$$\arg\min_{\Theta} \rho(\Theta) := \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\max_{\delta\in\mathcal{S}} L(\Theta, x+\delta, y)\right], \tag{6}$$

where $\Theta$ is the set of model parameters, $x$ is the input sample, $y$ is the label, $\mathcal{D}$ is the data distribution, $\delta$ is the perturbation, $\mathcal{S}$ is the perturbation space, $L$ is the loss function, and $\mathbb{E}$ is the empirical risk. The inner maximization problem attacks the neural network, while the outer minimization problem finds model parameters to make it robust against these adversarial attacks. In this work, we introduce a min-max formulation for the GSD problem, known as AGSD.

**Graph Diffusion Convolution (GDC).** Graph diffusion (Gasteiger et al., 2019) is defined via the diffusion matrix:

$$\mathbf{S} = \sum_{k=0}^{\infty} \theta_k \mathbf{T}^k, \tag{7}$$

where $\theta_k$ are the weighting coefficients and $\mathbf{T}$ is the transition matrix. Existing GNN architectures such as APPNP (Klicpera et al., 2019), GLP (Li et al., 2019), $S^2$GC (Zhu & Koniusz, 2021), and AirGNN (Liu et al., 2021a) can be viewed as variations of GDC. In this work, we introduce a new GDC architecture based on our proposed AGSD.

## 3. Graph Adversarial Diffusion Convolution

In this section, we first introduce the min-max variant of the graph signal denoising (GSD) problem in Section 3.1.1. Then, we discuss the closed-form solution for the inner maximization problem in Section 3.1.2. In Section 3.1.3, we propose graph adversarial diffusion convolution (GADC) by solving the outer minimization problem. Finally, Sections 3.2, 3.3, and 3.4 introduce four adaptive GADC variants designed to enhance robustness against adversarial attacks on the graph structure and noise in node features, as well as improve performance on heterophilic graphs.

### 3.1. Adversarial Graph Signal Denoising Problem

#### 3.1.1. Problem Formulation

Adversarial training has been recently explored in the robust optimization. Madry et al. (2018) use a saddle point (min-max) formulation to incorporate protection against adversarial attacks into neural networks. In the inner maximization problem, the adversary uses projected gradient descent (PGD) to identify the worst-case adversarial perturbations that maximize the loss. The outer minimization problem seeks to find model parameters that minimize the adversarial loss generated by the inner attack problem.

Inspired by the saddle point (min-max) formulation in adversarial training, we introduce the Adversarial Graph Signal Denoising (AGSD) problem as shown in Eq. (2). In the AGSD problem, the first term ensures the solution is close to the observed data, while the second term involves a graph Laplacian matrix. Defining perturbations on the graph structure to maximize the second term in the AGSD problem is less straightforward than traditional adversarial attacks. Inspired by the recent work (Lin et al., 2022) that uses spectral distance to deploy graph structural attacks, we leverage Laplacian distance to introduce perturbations.

**Laplacian Distance.** Lin et al. (2022) define the spectral distance as the changes in the eigenvalues of the graph Laplacian matrix, formulated as follows:

$$\mathcal{D}_{\text{spectral}} = \left\| g_\phi^*(\mathbf{\Lambda}) - g_\phi^*(\mathbf{\Lambda}') \right\|_2, \tag{8}$$

where $\mathbf{\Lambda}$ and $\mathbf{\Lambda}'$ denote the eigenvalues of the normalized graph Laplacian matrix for the original graph $\mathcal{G}$ and the disrupted graph $\mathcal{G}'$. The idea behind using the spectral distance for graph structural attacks is that perturbing the graph to maximize this distance induces the most harmful perturbation to the graph filters (Chang et al., 2021) and significantly disrupts node embeddings. To make the attack simple and effective, we directly add perturbations to the graph filter. This allows us to define the Laplacian distance.

$$\mathcal{D}_{\text{Laplacian}} = \left\| g_\phi(\tilde{\mathbf{L}}) - g_\phi(\mathbf{L}') \right\|_F = \left\| \mathbf{L}' - \tilde{\mathbf{L}} \right\|_F, \tag{9}$$

where $\tilde{\mathbf{L}} = \mathbf{I}_n - \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$ denotes the normalized graph Laplacian matrix.

**AGSD.** Based on the Laplacian distance, we introduce our AGSD as follows:

$$\begin{aligned} \arg\min_{\mathbf{F}} q(\mathbf{F}) := & \left[ \|\mathbf{F} - \mathbf{X}\|_F^2 + \lambda \cdot \max_{\mathbf{L}'} \operatorname{tr}\left(\mathbf{F}^\top \mathbf{L}' \mathbf{F}\right) \right] \\ & \text{s.t.} \left\| \mathbf{L}' - \tilde{\mathbf{L}} \right\|_F \le \varepsilon. \end{aligned} \tag{10}$$

In the inner optimization problem, a hypothetical adversary generates perturbations based on the Laplacian distance to create a modified Laplacian matrix, $\mathbf{L}'$, which aims to maximize the second term of the AGSD. Then, the outer minimization problem seeks to find $\mathbf{F}$ that minimizes the overall loss function $q(\mathbf{F})$.

### 3.1.2. Closed-form Solution of the Inner Maximization Problem

The min-max formulation in Eq.(10) introduces a more complex GSD problem. In contrast to adversarial training (Madry et al., 2018), where the inner maximization problem is solved using PGD before solving the outer minimization problem at each training step, our inner maximization problem is a quadratic optimization problem. This removes the need for computationally expensive adversarial perturbations at each step. Instead, we can directly identify the largest perturbation.

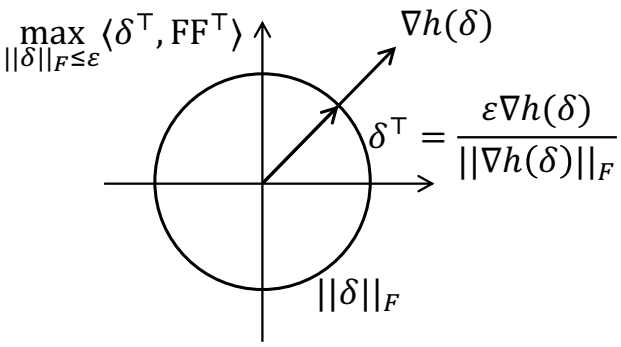



Figure 1: Illustration of the inner maximization problem. The loss function reaches the largest value when the direction of $\boldsymbol{\delta}^\top$ is the same as $\nabla h(\boldsymbol{\delta})$.

Let us denote the perturbations as $\boldsymbol{\delta}$, and the perturbed Laplacian matrix as $\mathbf{L}' = \tilde{\mathbf{L}} + \boldsymbol{\delta}$. Due to the quadratic nature of our inner maximization problem, we can reformulate it as follows:

$$\begin{aligned} \max_{\mathbf{L}'} \operatorname{tr}\left(\mathbf{F}^\top \mathbf{L}' \mathbf{F}\right) &= \langle \tilde{\mathbf{L}}^\top, \mathbf{F}\mathbf{F}^\top \rangle + \max_{\boldsymbol{\delta}} \langle \boldsymbol{\delta}^\top, \mathbf{F}\mathbf{F}^\top \rangle \\ & \text{s.t. } \|\boldsymbol{\delta}\|_F \le \varepsilon. \end{aligned} \tag{11}$$

We denote $h(\boldsymbol{\delta}) = \langle \boldsymbol{\delta}^\top, \mathbf{F}\mathbf{F}^\top \rangle$. $h(\boldsymbol{\delta})$ reaches the largest value when $\boldsymbol{\delta}^\top$ has the same direction with the gradient of $h(\boldsymbol{\delta})$, i.e. $\boldsymbol{\delta}^\top = \frac{\varepsilon \nabla h(\boldsymbol{\delta})}{\|\nabla h(\boldsymbol{\delta})\|_F} = \frac{\varepsilon \mathbf{F}\mathbf{F}^\top}{\|\mathbf{F}\mathbf{F}^\top\|_F}$. An illustration is provided in Figure 1. Plugging this solution into Eq. (10), we can rewrite the outer optimization problem as follows:

$$\begin{aligned} & \arg\min_{\mathbf{F}} q(\mathbf{F}), \\ q(\mathbf{F}) = & \left[ \|\mathbf{F} - \mathbf{X}\|_F^2 + \lambda \operatorname{tr}\left(\mathbf{F}^\top \tilde{\mathbf{L}} \mathbf{F}\right) + \lambda\varepsilon \operatorname{tr} \frac{\mathbf{F}^\top \mathbf{F}\mathbf{F}^\top \mathbf{F}}{\|\mathbf{F}\mathbf{F}^\top\|_F} \right], \end{aligned} \tag{12}$$

where we introduce an extra loss term $\lambda\varepsilon \operatorname{tr} \frac{\mathbf{F}^\top \mathbf{F}\mathbf{F}^\top \mathbf{F}}{\|\mathbf{F}\mathbf{F}^\top\|_F}$ compared with the GSD.

### 3.1.3. Modified Transition Matrix Derived by Outer Minimization Problem

In this section, we derive our graph adversarial diffusion convolution. By taking the gradient of Eq. (12) to zero, we get the solution of the outer optimization problem as:

$$\mathbf{F} = \left( \mathbf{I} + \lambda \tilde{\mathbf{L}} + \lambda \frac{\varepsilon \mathbf{F}\mathbf{F}^\top}{\|\mathbf{F}\mathbf{F}^\top\|_F} \right)^{-1} \mathbf{X}. \tag{13}$$

Computing Eq. (13) directly involves a matrix inverse operation, resulting in a complexity of $\mathcal{O}\left(n^3\right)$. This high computational cost can be prohibitively expensive for large

graphs. Inspired by the closed-form solution of Personalized PageRank (PPR) kernel (Brin, 1998; Gasteiger et al., 2019) (graph diffusion), we can solve Eq. (13) via graph diffusion (Eq. (7)) and obtain

$$\mathbf{F} = \frac{1}{\lambda+1}\sum_{k=0}^{K}\left[\frac{\lambda}{\lambda+1}\left(\tilde{\boldsymbol{\mathcal{A}}} - \frac{\varepsilon\mathbf{F}\mathbf{F}^\top}{\|\mathbf{F}\mathbf{F}^\top\|_F}\right)\right]^k \mathbf{X}, \quad (14)$$

where $\mathbf{T} = \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2} - \frac{\varepsilon\mathbf{F}\mathbf{F}^\top}{\|\mathbf{F}\mathbf{F}^\top\|_F} = \tilde{\boldsymbol{\mathcal{A}}} - \frac{\varepsilon\mathbf{F}\mathbf{F}^\top}{\|\mathbf{F}\mathbf{F}^\top\|_F}$ represents the transition matrix, and $\alpha = \frac{1}{\lambda+1}$ denotes the coefficient.

**Approximate Solution of Eq. (14).** Although we approximate the inverse matrix with graph diffusion to avoid $\mathcal{O}(n^3)$ complexity, the matrix series Eq. (14) still involves high powers of the matrix $\mathbf{F}$. Obviously, it is impossible to get an analytical solution of $\mathbf{F}$ for this equation. Current methods exploit iterative algorithms such as Newton's method (Moré & Sorensen, 1982) to solve this high-order nonlinear system of equations, but the computation is also very expensive. Therefore, there is no suitable solution that strictly follows the equation while also avoiding large computational complexity. We need a more efficient algorithm to solve Eq. (14). We observe that the first term of AGSD requires $\mathbf{F}$ to be close to $\mathbf{X}$, thus a natural idea is to replace $\mathbf{F}$ with $\mathbf{X}$ in the right side of Eq. (14) to reduce the computational complexity and improve the scalability for large graphs. Thus, we now formally propose our GADC as follows:

$$\mathbf{S} = \frac{1}{\lambda+1}\sum_{k=0}^{K}\left[\frac{\lambda}{\lambda+1}\mathbf{T}\right]^k, \quad (15)$$

where $\mathbf{T} = \tilde{\boldsymbol{\mathcal{A}}} - \frac{\varepsilon\mathbf{X}\mathbf{X}^\top}{\|\mathbf{X}\mathbf{X}^\top\|_F}$. Note that the additional term $\frac{\varepsilon\mathbf{X}\mathbf{X}^\top}{\|\mathbf{X}\mathbf{X}^\top\|_F}$ is a unique component introduced by the closed-form solution of the inner maximization problem of AGSD. This term doesn't appear in existing GDC architectures derived from the GSD problem. We will use this term to produce more adaptive architectures, as demonstrated in the following sections.

**Error Analysis.** We plug our obtained $\mathbf{F}$ into Eq. (14) to compute the error of this approximate solution for the equation on the Cora (Sen et al., 2008) dataset. The error matrix is computed by subtracting the left side from the right side: $\mathbf{F} - \frac{1}{\lambda+1}\sum_{k=0}^{K}\left[\frac{\lambda}{\lambda+1}\mathbf{T}\right]^k\mathbf{X}$, where $\mathbf{T} = \tilde{\boldsymbol{\mathcal{A}}} - \frac{\varepsilon\mathbf{X}\mathbf{X}^\top}{\|\mathbf{X}\mathbf{X}^\top\|_F}$. The norm of the error matrix for our solution is 4.5. As a comparison, we generate a random $\mathbf{F}$ from a Gaussian distribution with mean=1, std=1. If we use this random $\mathbf{F}$, the norm of the error matrix is 2780.6. Comparing the norm of the error matrix between our solution (4.5) and a random solution (2780.6), we can conclude that replacing $\mathbf{F}$ with $\mathbf{X}$ indeed achieves an accurate approximation, while greatly reducing the computational complexity.

**Scalability.** To leverage the additional term across various graphs, we enhance the scalability of the transition matrix by providing the following options: computing the normalized or unnormalized inner product of feature vectors $(\mathbf{X}_i, \mathbf{X}_j \mid j \in \mathcal{N}_i)$ between adjacent neighbors, or between every pair of nodes, similar to the masked/unmasked attention mechanism in GAT. Therefore, our modified transition matrix can be formulated as follows:

$$\begin{aligned}
&\mathbf{T} = \tilde{\boldsymbol{\mathcal{A}}} - \varepsilon\boldsymbol{\Phi}, \text{ where} \\
&\text{(I): } \Phi_{ij} := \begin{cases} \frac{\mathbf{X}_i\mathbf{X}_j^\top}{\|\mathbf{X}_i\|_2\|\mathbf{X}_j\|_2}, & \text{if } (v_i, v_j) \in \mathcal{E} \\ 0, & \text{otherwise,} \end{cases}; \\
&\text{(II): } \Phi_{ij} := \frac{\mathbf{X}_i\mathbf{X}_j^\top}{\|\mathbf{X}\mathbf{X}^\top\|_F}; \\
&\text{(III): } \Phi_{ij} := \begin{cases} \frac{\mathbf{X}_i\mathbf{X}_j^\top}{\|\mathbf{X}\mathbf{X}^\top\|_F}, & \text{if } (v_i, v_j) \in \mathcal{E} \\ 0, & \text{otherwise,} \end{cases}.
\end{aligned} \quad (16)$$

For the first option, we normalize the weights by using the product of the norms of the features of connected neighbors, which measures the similarity between these neighboring nodes' features. We will derive the fourth option from the first option to defend against adversarial attacks on the graph structure. The second option improves graph connectivity, filtering out large noise in graphs. Furthermore, we discover that even without explicitly incorporating additional graph connectivity (the third option), reassigning edge weights through the additional term improves denoising performance on large graphs. Moreover, using the first option to penalize the weights of connected neighbors enhances performance on heterophilic graphs. We will discuss the details of these options in the subsequent sections.

**Connection to APPNP.** If we take the limit $k \to \infty$ of power iterations in APPNP (Klicpera et al., 2019), APPNP converges to $\mathbf{Z}^{(\infty)} = \alpha\left(\mathbf{I} - (1-\alpha)\tilde{\boldsymbol{\mathcal{A}}}\right)^{-1}\mathbf{Z}^{(0)}$, similar to Eq. (13). Our proposed GADC introduces an additional term $\varepsilon\boldsymbol{\Phi}$ in the transition matrix. This additional term makes GADC more adaptable compared to APPNP. As we will demonstrate in the upcoming sections, this adaptability enhances resilience in various scenarios, including adversarial attacks on the graph structure, noise in node features, and inconsistent edges on heterophilic graphs.

### 3.2. Defending Against Graph Adversarial Attacks

Recent studies (Zügner et al., 2018; Zügner & Günnemann, 2019a) have shown that GNNs are vulnerable to adversarial attacks on the graph structure. These attacks often involve adding harmful edges or removing informative ones to manipulate the information flow, thus preventing GNNs from aggregating valuable information and disrupting node representations. GDC relies on the graph structure to derive the

diffusion matrix needed for smoothing node representations. However, when the graph structure is disrupted by adversarial attacks, it becomes unreliable, and we cannot use the graph Laplacian matrix to aggregate information from neighbors. Inspired by GNNGuard (Zhang & Zitnik, 2020), which assigns higher weights to edges connecting nodes with similar features while pruning edges between unrelated nodes, we employ the additional term to achieve a similar effect. The rationale is that on homophily graphs, nodes within the same class have similar features, while nodes from different classes exhibit dissimilar features. Therefore, we use the first option in Eq. (16) and assign a very large value to the additional term in the modified transition matrix, allowing this term to determine the edge weights:

$$\mathbf{T} = \tilde{\boldsymbol{\mathcal{A}}} - \lim_{\varepsilon \to \infty} \varepsilon \boldsymbol{\Phi}. \tag{17}$$

However, this approach may result in a numerical explosion problem. To address this issue, we introduce a trick in which we compute the modified transition matrix based on the additional term with $\mathbf{T} = -\lim_{\varepsilon\to\infty} \frac{1}{\varepsilon}\left(\tilde{\boldsymbol{\mathcal{A}}} - \varepsilon\boldsymbol{\Phi}\right)$. As such, we employ the cosine value to evaluate the similarity among connected neighbors and recalculate the weights of the transition matrix as follows:

$$\text{(IV):}\ \mathbf{T}_{ij} = (\mathbf{X}_i \odot \mathbf{X}_j) \,/\, \left(\|\mathbf{X}_i\|_2 \|\mathbf{X}_j\|_2\right)\ \text{s.t.}\ \mathbf{A}_{ij} = 1, \tag{18}$$

where $\mathbf{A}$ is the disrupted adjacency matrix by adversarial attacks and $\odot$ denotes the inner product. By implementing this trick, we can propose the fourth option to reconstruct the informative adjacency matrix using the additional term, thereby restoring the beneficial information flow during the aggregation process.

### 3.3. Tackling Large Noise in Node Features

In the study by Liu et al. (2021a), feature aggregation in GNNs is suggested to function as a low-pass filter, smoothing node features across neighborhoods and filtering out node feature noise (Nt & Maehara, 2019; Zhao & Akoglu, 2019). However, on graphs with a limited number of neighbors, GDC's ability to effectively filter out large noise in node features is constrained. To address this, we provide a theoretical analysis to understand the effect and introduce two simple and effective GADC (II, III) methods to handle noisy node features in graphs.

#### 3.3.1. Convergence Analysis for the Aggregated Noisy Matrix

Consider applying GDC to a noisy node feature matrix, represented as the product of $\mathbf{S}$ and $\boldsymbol{\Upsilon}$:

$$\mathbf{S}\boldsymbol{\Upsilon}, \tag{19}$$

where $\boldsymbol{\Upsilon}$ denotes the noise matrix. Intuitively, if the matrix norm $\|\mathbf{S}\boldsymbol{\Upsilon}\|_F$ can converge to a small value, the denoising effect is achieved. Consider the noise utilized in (Zhou et al., 2021; Chen et al., 2021; Zhang et al., 2022) follows Gaussian distribution, which is also covered by sub-Gaussian variable. Therefore, the noise matrix has the following property. Based on this property, we provide the analysis.

**Proposition 1 (Noise Property).** *Each entry of the noise matrix $\boldsymbol{\Upsilon}$, i.e., $[\boldsymbol{\Upsilon}]_{ij}$ is i.i.d sub-Gaussian random variable with variance $\sigma$ and mean $\mu = 0$, i.e.,*

$$\mathbb{E}\left[e^{\lambda([\boldsymbol{\Upsilon}]_{ij}-\mu)}\right] \le e^{\sigma^2\lambda^2/2} \quad \textit{for all } \lambda \in \mathbb{R}. \tag{20}$$

**Higher-order Graph Connectivity Factor.** Intuitively, $\mathbf{S}$ captures not only the connectivity of the graph structure (represented by $\tilde{\boldsymbol{\mathcal{A}}}$), but also the higher-order connectivity (represented by $\tilde{\boldsymbol{\mathcal{A}}}^2, \tilde{\boldsymbol{\mathcal{A}}}^3, \ldots, \tilde{\boldsymbol{\mathcal{A}}}^K$). As we will discuss later, greater higher-order graph connectivity can accelerate the convergence of the noise matrix. To formally quantify higher-order graph connectivity, we provide the following definition:

$$\tau = \max_i \tau_i, \quad \text{where } \tau_i = n \sum_{j=1}^{n} [\mathbf{S}]_{ij}^2 \Big/ \left([\mathbf{S}]_{ij}\right)^2. \tag{21}$$

**Remark 1.** *Here, we provide some intuition on why Eq. (21) represents higher-order graph connectivity. Note that each element in $\mathbf{S}$ is non-negative, and each row sum satisfies* [1]

$$\sum_{j=1}^{n} [\mathbf{S}]_{ij} = 1 - \left(\frac{\lambda}{\lambda+1}\right)^{K+1} = \beta. \tag{22}$$

*Based on Eq. (22), the sum of squares of elements in each row satisfies:*

$$\beta^2 \Big/ n \le \sum_{j=1}^{n} [\mathbf{S}]_{ij}^2 \le \beta^2. \tag{23}$$

*When a graph has high connectivity, meaning the elements in row $i$ of $\boldsymbol{S}$ are more uniformly distributed, Eq. (23) reaches its lower bound. Conversely, if a graph is poorly connected and only one element in row $i$ is greater than zero, Eq. (23) reaches its upper bound. Therefore, the value of $\tau \in [1, n]$ is determined as follows: when the higher-order graph connectivity factor is large, $\tau \to 1$, and when the graph is less connected, $\tau \to n$.*

**Theorem 1 (Upper Bound).** *Suppose we choose $t = 2\tau\left(4\log n + \log 2d\right)/n$. Then, with a high probability of $1 - 1/d$, we have*

$$\|\mathbf{S}\boldsymbol{\Upsilon}\|_F^2 \le \frac{2\tau\left(1 - \left(\frac{\lambda}{\lambda+1}\right)^{K+1}\right)^2 \sigma^2\left(4\log n + \log 2d\right)}{n}. \tag{24}$$

[1] This result is obtained by using $\tilde{\boldsymbol{\mathcal{A}}} = \tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}$ for ease of theoretical analysis, while in experiments we use the more common $\tilde{\boldsymbol{\mathcal{A}}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$ as suggested in GCN. The proof can be found in Appendix A.

Proof can be found in Appendix B. Theorem 1 implies that the norm of the aggregated noise matrix $\mathbf{S}\boldsymbol{\Upsilon}$ is bounded by three terms: the number of nodes of a graph $n$, the expansion order $K$, the higher-order graph connectivity factor $\tau$. We provide the intuition behind Theorem 1 by viewing $\mathbf{S}\boldsymbol{\Upsilon}$ as sampling from the noise probability distribution. Each row of $\mathbf{S}$ represents the weighted average of these noise samples. According to the law of large numbers, each row of $\mathbf{S}\boldsymbol{\Upsilon}$ will converge to the expected value ($\mu = 0$) of the noise distribution, thus reducing the non-predictive stochasticity of the weighted average noise. The denoising effect of $\mathbf{S}\boldsymbol{\Upsilon}$ depends on the number of samples (i.e., $K$ ) and the weights $\mathbf{S}_{ij}$, both of which are influenced by the depth of GNNs and the graph structure. We provide an illustration in Appendix H.

#### 3.3.2. Architecture

For low node-degree graphs, it is impossible to increase the node degree; however, introducing the additional term $\frac{\varepsilon \mathbf{X}\mathbf{X}^\top}{\|\mathbf{X}\mathbf{X}^\top\|_F}$ can add additional graph connectivity, thereby decreasing the higher-order graph connectivity factor. Thus, we employ the second option in Eq. (16) for low node-degree noisy graphs. Furthermore, considering the noise comes from a Gaussian distribution, the magnitude of the added noise values can be unpredictable. Nonetheless, the term $\left\|\mathbf{X}\mathbf{X}^\top\right\|_F$ can penalize the weights when noise values are excessively large, thereby mitigating the adverse effect of the noise. Note that the weights of edges also determine the factor as shown in Eq. (21). For high node-degree graphs, we can reassign the weights of edges through the additional term to decrease the factor as shown in the third option. In the experiment section, we will demonstrate the effectiveness of our GADC (II, III) in dealing with noise in node features for both types of graphs.

### 3.4. Improving Performance on Heterophilic Graphs

The aggregation scheme in message-passing-based GNNs is generally considered harmful for learning node representations on heterophilic graphs. To alleviate the negative impact of inconsistent edges on heterophilic graphs, we employ the first option in Eq. (16) to reduce the weights of the Laplacian matrix coefficients. Thus, the additional term, $\varepsilon\boldsymbol{\Phi}$, can obstruct the graph smoothing process. In our subsequent ablation study, we will demonstrate our GADC (I) can improve performance on heterophilic graphs.

### 3.5. Decoupling Feature Aggregation from Downstream Training

GADC is formulated as Eq. (15), with four options presented in Eq. (16) and Eq. (18). Depending on the scenario, we first select the appropriate option and then calculate $\Phi_{ij}$ using the feature matrix $\mathbf{X}$. This allows us to obtain the graph diffusion matrix $\mathbf{S}$. We then multiply the graph diffusion matrix by the feature matrix $\mathbf{X}$ to get the aggregated features $\mathbf{S}\mathbf{X}$. Finally, we use the aggregated features as input to train a linear model or an MLP. This approach effectively decouples feature aggregation from downstream training. The overall process is illustrated in Algorithm 1.

**Algorithm 1** Graph Adversarial Diffusion Convolution

**Input:** Adjacency matrix $\mathbf{A}$, feature matrix $\mathbf{X}$
**Output:** Prediction $\mathbf{Z}$
Compute the transition matrix $\mathbf{T}$ using Eq. (16) based on the selected option.
Compute the graph adversarial diffusion matrix $\mathbf{S}$ via Eq. (15).
Compute the aggregated feature matrix $\mathbf{F} = \mathbf{S}\mathbf{X}$.
**while** not convergence **do**
  Compute the output of the forward model (Linear or MLP) based on the input $\mathbf{F}$: $\mathbf{Z} = f_\Theta(\mathbf{F})$.
  Compute the supervised loss function $\mathcal{L}_s$.
  Update the parameters $\Theta$ via gradient descent: $\Theta = \Theta - \eta\nabla_\Theta(\mathcal{L}_s)$.
**end while**
Predict via $\mathbf{Z} = f_\Theta(\mathbf{F})$.

**Differences from APPNP and GNNGuard.** For the fourth option, our algorithm is a pre-computation method that uses the additional term to reconstruct the graph structure once. After reconstructing the adjacency matrix, we use it directly to aggregate node features. GNNGuard evaluates the importance of neighbors using the cosine similarity of node embeddings for each layer and normalizes the similarity. Therefore, the features used for computing similarity differ between GADC (I) and the higher layers ($\geq 2$) of GNNGuard. Additionally, our GADC differs from APPNP. APPNP performs a nonlinear transformation on the feature matrix and then uses the graph diffusion matrix to aggregate node features. In contrast, we introduce an additional term to create more adaptable GADC architectures.

## 4. Experiments

In this section, we conduct extensive experiments to demonstrate the effectiveness of our proposed GADC architectures in various scenarios.

### 4.1. Evaluation of Defense against Adversarial Attacks on the Graph Structure

In this section, we demonstrate the defense performance of our proposed GADC (IV) against adversarial attacks on the graph structure, including non-adaptive and adaptive attacks (Mujkanovic et al., 2022), by utilizing the additional term in the modified transition matrix. We conduct experi-

Table 1: Summary of test accuracy (%) results from 100 runs under non-adaptive adversarial attacks.

| Ptb Rate (%) | Cora | | | Citeseer | | | Pubmed | | |
|---|---|---|---|---|---|---|---|---|---|
| | 25 | 50 | 75 | 25 | 50 | 75 | 25 | 50 | 75 |
| GCN | 54.09 ± 1.48 | 33.55 ± 2.63 | 22.94 ± 3.06 | 56.21 ± 1.26 | 42.41 ± 2.82 | 32.00 ± 3.08 | 43.29 ± 4.75 | 35.91 ± 3.91 | 35.33 ± 4.31 |
| GAT | 57.70 ± 1.30 | 37.76 ± 3.21 | 20.01 ± 3.45 | 59.15 ± 1.03 | 46.74 ± 3.06 | 37.25 ± 1.32 | 30.37 ± 2.15 | 28.08 ± 3.80 | 26.09 ± 4.53 |
| APPNP | 56.08 ± 1.06 | 33.23 ± 1.48 | 15.61 ± 0.72 | 60.15 ± 1.07 | 50.95 ± 1.91 | 40.15 ± 1.14 | 38.36 ± 3.73 | 39.94 ± 0.00 | 39.94 ± 0.00 |
| $S^2GC$ | 56.02 ± 0.03 | 31.61 ± 0.03 | 20.89 ± 0.57 | 50.18 ± 0.00 | 34.12 ± 0.00 | 25.49 ± 0.03 | 31.59 ± 2.44 | 39.93 ± 0.02 | 39.94 ± 0.00 |
| NAGphormer | 62.11 ± 1.95 | 41.31 ± 2.41 | 28.84 ± 1.36 | 64.19 ± 1.01 | 56.29 ± 1.00 | 46.58 ± 1.15 | 71.49 ± 1.43 | 45.26 ± 2.58 | 36.76 ± 5.64 |
| Robust-GCN | 56.23 ± 0.70 | 36.87 ± 1.33 | 27.25 ± 2.23 | 56.67 ± 0.59 | 41.95 ± 0.94 | 30.69 ± 1.45 | 32.43 ± 1.40 | 33.42 ± 2.36 | 33.44 ± 2.58 |
| GCN-Jaccard | 65.70 ± 1.03 | 46.31 ± 2.25 | 32.62 ± 1.31 | 60.28 ± 1.09 | 47.73 ± 1.66 | 38.68 ± 2.66 | 43.79 ± 4.75 | 35.89 ± 3.92 | 35.31 ± 4.31 |
| GCN-SVD | 58.27 ± 0.97 | 36.49 ± 1.86 | 25.13 ± 3.23 | 66.97 ± 0.84 | 59.21 ± 0.91 | 40.69 ± 1.10 | 78.69 ± 0.50 | 55.05 ± 0.93 | 36.31 ± 2.44 |
| Pro-GNN | 72.21 ± 1.89 | 38.86 ± 0.78 | 24.34 ± 3.06 | 67.18 ± 1.36 | 50.80 ± 2.05 | 31.60 ± 2.19 | - | - | - |
| GNNGuard | 54.43 ± 1.45 | 34.48 ± 2.45 | 25.98 ± 3.31 | 55.84 ± 1.68 | 41.23 ± 1.85 | 31.80 ± 1.62 | 44.56 ± 2.91 | 37.56 ± 4.28 | 37.45 ± 4.92 |
| Elastic GNN | 57.95 ± 3.28 | 45.18 ± 1.90 | 30.30 ± 2.91 | 64.24 ± 1.03 | 53.19 ± 2.74 | 42.96 ± 1.97 | 54.45 ± 0.60 | 39.94 ± 0.00 | 39.94 ± 0.00 |
| STABLE | **78.69 ± 0.50** | **71.94 ± 0.69** | 62.98 ± 1.03 | 70.49 ± 0.95 | 64.04 ± 2.07 | 52.03 ± 3.16 | 33.68 ± 5.38 | 36.57 ± 5.54 | 35.98 ± 6.37 |
| EvenNet | 76.30 ± 0.39 | 71.11 ± 0.46 | 67.07 ± 0.60 | 70.89 ± 0.71 | 66.20 ± 0.87 | 63.60 ± 1.71 | 83.98 ± 0.00 | 81.52 ± 0.55 | 81.27 ± 0.46 |
| GCN-GARNET | 74.80 ± 1.19 | 70.90 ± 1.19 | **67.49 ± 1.06** | 70.01 ± 0.96 | 63.76 ± 1.99 | 56.64 ± 2.88 | 85.14 ± 0.34 | 84.82 ± 0.47 | 84.74 ± 0.42 |
| HANG-quad | 67.54 ± 1.03 | 65.22 ± 1.07 | 61.88 ± 1.50 | 66.37 ± 0.95 | 65.31 ± 0.93 | 63.68 ± 1.32 | 85.03 ± 0.20 | 85.06 ± 0.20 | 84.89 ± 0.17 |
| GADC (IV) | 76.00 ± 0.61 | 70.29 ± 0.82 | 66.06 ± 0.66 | **71.55 ± 0.89** | **66.47 ± 0.99** | **65.25 ± 0.69** | **86.74 ± 0.21** | **85.92 ± 0.14** | **85.29 ± 0.06** |

Table 2: Summary of test accuracy (%) results from one run under adaptive adversarial attacks.

| Dataset | Evasion Test Accuracy | | Poisoned Test Accuracy | |
|---|---|---|---|---|
| | Cora | Citeseer | Cora | Citeseer |
| GCN | 59.71 | 62.78 | 48.99 | 46.98 |
| SVD-GCN | 57.44 | 58.25 | 45.79 | 49.41 |
| GNNGuard | 66.35 | 66.50 | 51.07 | 49.70 |
| Soft-Median-GDC | 67.05 | 63.88 | 56.81 | 58.18 |
| GADC (IV) | **72.03** | **72.94** | **70.73** | **68.36** |

ments on three citation network datasets (Sen et al., 2008): Cora, Citeseer, and Pubmed. The statistics for all datasets used in this paper can be found in Appendix D. For non-adaptive attacks, we use Mettack (Zügner & Günnemann, 2019a), and for adaptive attacks, we use Aux-Attack.

**Baselines.** For the baselines on non-adaptive attacks, we use two popular GNNs: GCN (Kipf & Welling, 2017) and GAT (Veličković et al., 2018); two GDCs: APPNP (Klicpera et al., 2019) and $S^2GC$ (Zhu & Koniusz, 2021); a graph transformer: NAGphormer (Chen et al., 2023); and various defense methods against Meta-attack (Zügner & Günnemann, 2019a), including Robust-GCN (Zügner & Günnemann, 2019b), GCN-Jaccard (Wu et al., 2019b), GCN-SVD (Entezari et al., 2020), Pro-GNN (Jin et al., 2020), GNNGuard (Zhang & Zitnik, 2020), Elastic GNN (Liu et al., 2021b), STABLE (Li et al., 2022), EvenNet (Lei et al., 2022), GCN-GARNET (Deng et al., 2022), and HANG-quad (Zhao et al., 2023). For the baselines on adaptive attacks, we consider GCN, GCN-SVD, GNNGuard, and Soft-Median-GDC (Geisler et al., 2021).

**Setting.** For non-adaptive attacks, we use DeepRobust (Li et al., 2020) to generate disrupted graph structures on Cora, Citeseer, and Pubmed datasets with perturbation rates of $0.25, 0.5$, and $0.75$. We follow dataset splits used in Jin et al. (2020). Each experiment is repeated 10 times, and we report the mean test accuracy and its standard deviation on the node classification task. For our GADC (IV) model, we add a 2-layer MLP with 32 hidden units after feature aggregation. We tune hyper-parameters with the validation dataset. We also tune some baselines such as HANG-quad, GCN-GARNET, and EvenNet. All hyper-parameter details of our methods can be found in Appendix E. For adaptive attacks, we set the attack budget to 1000 for the Cora and Citeseer datasets and use dataset splits from Mujkanovic et al. (2022). The experiment is repeated once. For our model, we set $K = 2$ and discard the terms for $k = 0/1$, similar to SGC (Wu et al., 2019a).

**Results.** Table 1 reports the results of non-adaptive attacks for our method and other baselines. Our method outperforms other baselines on Citeseer and Pubmed and achieves performance relatively close to the best baseline on Cora. Additionally, our GADC (IV) significantly outperforms both APPNP and $S^2GC$. As the perturbation rate increases, the adjacency matrix becomes increasingly unreliable. However, our additional term leverages the inherent homophily properties of these graphs to reconstruct the clean adjacency matrix, thereby restoring effective information flow within the aggregation scheme. GNNGuard and NAGphormer also evaluate the importance of neighbors using the attention mechanism on node embeddings at each layer to mitigate the disruption of the graph structure caused by adversarial attacks. In contrast, our algorithm is a pre-computation method that uses the additional term to reconstruct the graph structure directly in one step. After reconstructing the adjacency matrix, we use it directly to aggregate features. We think that smoothing node features makes the recalculated weights less accurate. Therefore, our method performs much better than GNNGuard and NAGphormer.

Table 3: Summary of test accuracy (%) results from 100 runs on citation network datasets with Gaussian noise.

| Cora | MLP | GCN | GAT | APPNP | GLP | $S^2$GC | IRLS | AirGNN | GADC (II) |
|---|---|---|---|---|---|---|---|---|---|
| 0.1 | 41.0 ± 9.1 | 53.5 ± 25.1 | 73.9 ± 8.7 | 78.1 ± 8.7 | 65.5 ± 13.9 | 74.0 ± 10.5 | 65.8 ± 12.9 | **78.8 ± 6.9** | 77.4 ± 2.5 |
| 0.2 | 21.6 ± 6.5 | 41.3 ± 20.7 | 62.8 ± 12.7 | 72.1 ± 10.3 | 55.4 ± 12.4 | 65.4 ± 12.1 | 47.8 ± 8.9 | **73.6 ± 9.4** | 72.6 ± 2.8 |
| 0.3 | 17.5 ± 6.4 | 32.8 ± 13.6 | 51.2 ± 14.1 | 66.3 ± 12.4 | 49.9 ± 11.1 | 60.8 ± 10.9 | 42.3 ± 7.7 | 68.5 ± 11.5 | **69.1 ± 3.2** |
| 0.4 | 15.9 ± 6.0 | 30.0 ± 10.0 | 45.4 ± 12.3 | 62.3 ± 12.9 | 49.4 ± 10.0 | 55.9 ± 11.8 | 40.9 ± 7.3 | 65.6 ± 11.3 | **68.0 ± 3.6** |
| 0.5 | 14.9 ± 5.2 | 28.1 ± 9.1 | 41.9 ± 13.3 | 63.0 ± 11.8 | 47.3 ± 11.7 | 53.4 ± 11.4 | 41.4 ± 7.3 | **67.6 ± 8.0** | **67.6 ± 3.3** |
| 100 | 15.0 ± 4.6 | 28.4 ± 7.5 | 31.9 ± 12.9 | 61.8 ± 11.2 | 47.6 ± 10.3 | 52.0 ± 13.1 | 43.7 ± 6.6 | 65.4 ± 11.4 | **66.9 ± 5.3** |
| **Citeseer** | **MLP** | **GCN** | **GAT** | **APPNP** | **GLP** | **$S^2$GC** | **IRLS** | **AirGNN** | **GADC (II)** |
| 0.1 | 46.3 ± 3.1 | 52.4 ± 21.9 | 69.5 ± 1.1 | 70.3 ± 1.0 | 65.3 ± 3.0 | 71.7 ± 1.1 | 70.8 ± 3.3 | **70.9 ± 1.3** | 69.6 ± 1.2 |
| 0.2 | 25.0 ± 5.2 | 37.3 ± 15.9 | 55.1 ± 10.4 | **59.6 ± 9.6** | 47.2 ± 8.4 | 59.5 ± 7.3 | 56.0 ± 7.2 | 58.9 ± 13.9 | 59.5 ± 3.5 |
| 0.3 | 17.9 ± 3.2 | 24.4 ± 4.4 | 36.2 ± 9.8 | 45.9 ± 11.7 | 36.4 ± 7.1 | 46.6 ± 8.9 | 37.2 ± 7.0 | 44.2 ± 14.9 | **50.5 ± 3.1** |
| 0.4 | 17.4 ± 3.0 | 23.3 ± 4.4 | 30.9 ± 7.1 | 40.8 ± 10.4 | 36.7 ± 4.7 | 42.7 ± 6.7 | 36.3 ± 4.9 | 39.8 ± 12.3 | **48.3 ± 2.3** |
| 0.5 | 17.4 ± 2.4 | 22.7 ± 4.0 | 28.7 ± 6.2 | 39.5 ± 8.8 | 36.4 ± 4.8 | 41.3 ± 6.8 | 36.7 ± 5.0 | 36.9 ± 12.1 | **47.8 ± 2.4** |
| 100 | 16.8 ± 2.5 | 23.7 ± 3.7 | 25.0 ± 6.5 | 36.6 ± 9.4 | 38.0 ± 4.0 | 41.4 ± 5.3 | 37.3 ± 4.8 | 33.4 ± 11.4 | **46.9 ± 2.4** |
| **Pubmed** | **MLP** | **GCN** | **GAT** | **APPNP** | **GLP** | **$S^2$GC** | **IRLS** | **AirGNN** | **GADC (II)** |
| 0.01 | 67.9 ± 1.4 | 61.1 ± 18.2 | 77.7 ± 0.9 | **80.0 ± 0.7** | 78.6 ± 0.9 | 79.2 ± 0.5 | 81.2 ± 0.8 | 79.9 ± 0.4 | 77.4 ± 0.8 |
| 0.02 | 58.3 ± 2.0 | 61.0 ± 17.4 | 75.8 ± 1.3 | 78.3 ± 1.1 | 76.5 ± 1.1 | 78.0 ± 0.8 | 78.4 ± 1.1 | **79.3 ± 0.7** | 76.4 ± 1.0 |
| 0.03 | 47.9 ± 4.1 | 54.0 ± 14.2 | 60.7 ± 13.6 | 74.4 ± 4.9 | 70.3 ± 5.7 | 74.4 ± 2.5 | 69.4 ± 6.9 | **76.2 ± 4.8** | 73.9 ± 1.6 |
| 0.04 | 39.4 ± 4.8 | 41.7 ± 9.7 | 40.0 ± 7.2 | 64.1 ± 14.4 | 60.8 ± 8.4 | 62.7 ± 11.2 | 58.1 ± 8.7 | 65.9 ± 13.3 | **69.0 ± 2.3** |
| 0.05 | 36.5 ± 6.6 | 36.9 ± 6.7 | 38.4 ± 6.3 | 55.3 ± 15.9 | 56.9 ± 7.6 | 55.8 ± 10.9 | 55.9 ± 9.7 | 58.9 ± 14.3 | **66.3 ± 2.9** |
| 100 | 35.0 ± 5.3 | 36.6 ± 5.3 | 38.2 ± 3.7 | 52.3 ± 11.3 | 55.1 ± 5.6 | 54.0 ± 10.3 | 50.1 ± 10.6 | 55.6 ± 13.6 | **62.5 ± 3.6** |

Table 4: Summary of test accuracy (%) results from 10 runs on the Coauthor-CS and Coauthor-Phy datasets with Gaussian noise.

| Noise Level | Coauthor-CS | | Coauthor-Phy | |
|---|---|---|---|---|
| | 0.1 | 1 | 0.1 | 1 |
| MLP | 82.5±1.8 | 22.3±0.1 | 81.6±8.1 | 47.0±10.0 |
| GCN | 87.3±0.5 | 61.3±14.3 | 94.2±0.4 | 78.6±10.6 |
| GAT | 86.8±3.6 | 57.9±20.2 | 94.0±0.4 | 63.7±16.7 |
| APPNP | 94.5±0.4 | 81.7±2.0 | 95.4±0.3 | 89.2±1.6 |
| GLP | 91.3±0.4 | 52.4±17.3 | 93.3±2.5 | 81.3±10.6 |
| $S^2$GC | 86.1±0.2 | 79.6±10.2 | 92.6±1.3 | 89.4±4.3 |
| IRLS | 78.8±5.1 | 62.1±17.8 | 89.2±3.4 | 87.0±4.5 |
| GADC (II) | **95.4±0.2** | **87.8±1.5** | **95.7±0.2** | **93.6±0.8** |

Table 5: Summary of test accuracy (%) results from 10 runs on the ogbn-products dataset with Gaussian noise.

| Noise Level | ogbn-products | |
|---|---|---|
| | 0.1 | 1 |
| MLP | 59.68±0.16 | 38.08±0.10 |
| GCN | 75.60±0.19 | 72.76±0.20 |
| $S^2$GC | 74.95±0.13 | 63.17±0.12 |
| GADC (III) | **77.54±0.15** | **73.66±0.13** |

Table 2 reports the experimental results of our method and the baselines under adaptive attacks. The results demonstrate that our method significantly outperforms the baselines. This indicates that even when adaptive attacks modify the graph structure during inference, our additional terms can still effectively reconstruct the adjacency matrix and provide a robust defense.

### 4.2. Evaluation of Denoising against Noise in Node Features

In this section, we compare the denoising performance of our proposed GADC (II, III) with various baselines by evaluating their test accuracy when models are trained on noisy feature matrices.

**Datasets.** For our experiments, we use three small-scale graph datasets: Cora, Citeseer, and Pubmed, and three large-scale graph datasets: Coauthor-CS, Coauthor-Phy (Shchur et al., 2018), and ogbn-products (Hu et al., 2020). We follow dataset splits used in (Yang et al., 2016) for the citation network datasets. For the Coauthor datasets, we split the nodes into 60% for training, 20% for validation, and 20% for testing. For the ogbn-products dataset, we follow dataset splits provided by OGB (Hu et al., 2020).

**Baselines.** For the baselines, we consider several variants of GDC, including APPNP (Klicpera et al., 2019), GLP (Li et al., 2019), $S^2$GC (Zhu & Koniusz, 2021), and AirGNN (Liu et al., 2021a). Additionally, we include popular GNNs such as GCN (Kipf & Welling, 2017) and GAT (Veličković et al., 2018). We also consider IRLS (Yang et al., 2021), a GNN architecture derived from GSD, and an MLP, which does not employ any aggregation scheme.

**Setting.** We assume that the original feature matrix is clean and devoid of noise. To introduce noise, we synthesize it from a standard Gaussian distribution and add it to the original feature matrix, as suggested in AirGNN (Liu et al., 2021a). After adding Gaussian noise, we apply row normalization to node features and train all models

Table 6: Summary of ablation study results in terms of test accuracy (%).

| Dataset | Cora | | Coauthor-CS | | Coauthor-Phy | | ogbn-products | |
|---|---|---|---|---|---|---|---|---|
| Noise Level | 0.1 | 0.5 | 0.1 | 1.0 | 0.1 | 1.0 | 0.1 | 1.0 |
| GADC (II/III, $\varepsilon = 0$) | $76.4 \pm 3.2$ | $66.5 \pm 5.1$ | $95.3 \pm 0.2$ | $87.1 \pm 3.1$ | $95.7 \pm 0.2$ | $93.1 \pm 1.4$ | $77.5 \pm 0.2$ | $73.4 \pm 0.1$ |
| GADC (II/III, $\varepsilon \neq 0$) | $77.4 \pm 2.5$ | $67.6 \pm 3.3$ | $95.4 \pm 0.2$ | $87.8 \pm 1.5$ | $95.7 \pm 0.2$ | $93.6 \pm 0.8$ | $77.5 \pm 0.2$ | $73.7 \pm 0.1$ |

using these noisy feature matrices. The noise level $\xi$ controls the magnitude of the noise added to the feature matrix: $\mathbf{X} + \xi\boldsymbol{\Upsilon}$, where $\boldsymbol{\Upsilon}$ is sampled from a standard i.i.d. Gaussian distribution. For Cora and Citeseer, we test $\xi \in \{0.1, 0.2, 0.3, 0.4, 0.5, 100\}$, and for Pubmed, we test $\xi \in \{0.01, 0.02, 0.03, 0.04, 0.05, 100\}$. For Coauthor-CS, Coauthor-Phy, and ogbn-products, we test $\xi \in \{0.1, 1.0\}$. For each model's hyperparameters, we follow the settings reported in their original papers. To reduce randomness, we repeat the experiment 100 or 10 times and report the mean test accuracy. In each repeated run, different Gaussian noises are added; however, within the same run, the same noisy feature matrix is used to train all models. For the citation network datasets, we employ the second option in Eq. (16), and we set $K = 16$ and $\lambda = 32$ by default. We select $K$ for other datasets based on the best performance on the validation dataset. For coauthor datasets, we use the second option. For ogbn-products, we use the third option and don't add additional graph connectivity since it is a very large graph. Therefore, we only compute $\Phi_{ij}$ based on connected neighbors in the original graph.

**Results.** Table 3, 4, and 5 report test accuracy results across various noise levels for node classification tasks. As demonstrated in Table 3, 4, and 5, under high noise levels, the performance of MLP approximates random guessing, as the test accuracy is close to the inverse of the total number of labels. This implies that the added noise has effectively obscured the original features. Compared to GDC variants, our proposed GADC (II, III) demonstrates superior denoising performance at high noise levels. This demonstrates the effectiveness of the additional term in our modified transition matrix for tracking large noise. We also synthesize noise by flipping individual features with a small Bernoulli probability on three citation network datasets. We report the results in Appendix F.

**Computational Complexity.** Introducing additional edges in GADC (II) does increase the time complexity of the aggregation process. However, we mitigate this issue by decoupling the aggregation process from downstream training, ensuring that aggregation occurs only once. This differs from APPNP, which repeats the aggregation process during each training iteration (i.e., aggregation occurs $n$ times for $n$ iterations). As a result, completing 100 runs (one experiment) of GADC (II) on the Pubmed dataset takes only 58 seconds, and completing 10 runs on the Coauthor-CS dataset takes 22 seconds. Without introducing additional edges ($\varepsilon = 0$), 100 runs on the Pubmed dataset take only 41 seconds. In contrast, APPNP requires 232 seconds to complete 100 runs on the Pubmed dataset. This demonstrates the computational efficiency of our method. For the ogbn-products dataset, we maintain only the one-hop connections and adjust the edge weights, without introducing additional edges, resulting in a time complexity similar to that of GDC.

### 4.3. Ablation Study on the Effectiveness of the Additional Term

We conduct an ablation study by setting $\varepsilon$ to zero to observe its impact. As shown in Table 6, for the low node-degree graph (Cora), our additional term effectively enhances denoising performance under both small and large noise. However, for high node-degree graphs (Coauthor and ogbn-products), performance improvement is noticeable only under large noise. This is because low node-degree graphs have relatively fewer neighboring nodes, so introducing the additional term increases graph connectivity, effectively filtering out noise. In high node-degree graphs, the inherent connectivity is already strong enough under small noise, reducing the need for additional connectivity to filter out noise. However, under large noise, enhanced connectivity still improves denoising for Co-author. In the ogbn-products graph, although GADC (III, $\varepsilon \neq 0$) doesn't introduce new connections, it reassigns the weights of existing edges, successfully reducing the value of $\tau$ and thereby improving denoising performance.

### 4.4. Improving Performance on Heterophilic Graphs

We conduct an ablation study to show that our GADC (I) can improve performance on heterophilic graphs. Due to the space limit, we provide the experimental results in Appendix G and discuss related work in Appendix C.

## 5. Conclusion

We introduce a min-max formulation of the graph signal denoising problem and develop various GADC variants. Extensive results demonstrate that our GADC effectively addresses noisy features in graphs, graph structure attacks, and inconsistent edges on heterophilic graphs.

## Acknowledgements

We thank all the anonymous reviewers for their helpful comments and suggestions. Songtao Liu thanks ICML and NeurIPS so much for providing financial aid, making him attend ICML 2022/2023 and NeurIPS 2023 in person, as he was unable to get other funding sources for attendance (ICML 2023, NeurIPS 2023).

## Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

## References

Brin, S. The pagerank citation ranking: bringing order to the web. *Proceedings of ASIS, 1998*, 98:161–172, 1998.

Chang, H., Rong, Y., Xu, T., Bian, Y., Zhou, S., Wang, X., Huang, J., and Zhu, W. Not all low-pass filters are robust in graph convolutional networks. In *Advances in Neural Information Processing Systems*, 2021.

Chang, H., Rong, Y., Xu, T., Huang, W., Zhang, H., Cui, P., Wang, X., Zhu, W., and Huang, J. Adversarial attack framework on graph embedding models with limited knowledge. *IEEE Transactions on Knowledge and Data Engineering*, 35(5):4499–4513, 2022.

Chen, J., Gao, K., Li, G., and He, K. NAGphormer: A tokenized graph transformer for node classification in large graphs. In *International Conference on Learning Representations*, 2023.

Chen, Q., Wang, Y., Wang, Y., Yang, J., and Lin, Z. Optimization-induced graph implicit nonlinear diffusion. In *International Conference on Machine Learning*, 2022.

Chen, S., Sandryhaila, A., Moura, J. M., and Kovacevic, J. Signal denoising on graphs via graph filtering. In *2014 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pp. 872–876. IEEE, 2014.

Chen, S., Eldar, Y. C., and Zhao, L. Graph unrolling networks: Interpretable neural networks for graph signal denoising. *IEEE Transactions on Signal Processing*, 69: 3699–3713, 2021.

Dai, H., Li, H., Tian, T., Huang, X., Wang, L., Zhu, J., and Song, L. Adversarial attack on graph structured data. In *International Conference on Machine Learning*, 2018.

Dai, H., Li, C., Coley, C. W., Dai, B., and Song, L. Retrosynthesis prediction with conditional graph logic network. In *Advances in Neural Information Processing Systems*, 2019.

Deng, C., Li, X., Feng, Z., and Zhang, Z. Garnet: Reduced-rank topology learning for robust and scalable graph neural networks. In *Learning on Graphs Conference*, 2022.

Entezari, N., Al-Sayouri, S. A., Darvishzadeh, A., and Papalexakis, E. E. All you need is low (rank) defending against adversarial attacks on graphs. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, 2020.

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

Gasteiger, J., Weißenberger, S., and Günnemann, S. Diffusion improves graph learning. In *Advances in Neural Information Processing Systems*, 2019.

Geisler, S., Schmidt, T., Şirin, H., Zügner, D., Bojchevski, A., and Günnemann, S. Robustness of graph neural networks at scale. In *Advances in Neural Information Processing Systems*, 2021.

Gosch, L., Geisler, S., Sturm, D., Charpentier, B., Zügner, D., and Günnemann, S. Adversarial training for graph neural networks: Pitfalls, solutions, and new directions. In *Advances in Neural Information Processing Systems*, 2023.

Guo, S., Lin, Y., Feng, N., Song, C., and Wan, H. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, 2017.

Hoeffding, W. Probability inequalities for sums of bounded random variables. *The Collected Works of Wassily Hoeffding*, pp. 409–426, 1994.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

Jia, J. and Benson, A. R. A unifying generative model for graph learning algorithms: Label propagation, graph convolutions, and combinations. *SIAM Journal on Mathematics of Data Science*, 4(1):100–125, 2022.

Jin, W., Ma, Y., Liu, X., Tang, X., Wang, S., and Tang, J. Graph structure learning for robust graph neural networks. In *Proceedings of the 26th ACM SIGKDD International*

*Conference on Knowledge Discovery & Data Mining*, 2020.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

Klicpera, J., Bojchevski, A., and Günnemann, S. Predict then propagate: Graph neural networks meet personalized pagerank. In *International Conference on Learning Representations*, 2019.

Lei, R., Wang, Z., Li, Y., Ding, B., and Wei, Z. Evennet: Ignoring odd-hop neighbors improves robustness of graph neural networks. In *Advances in Neural Information Processing Systems*, 2022.

Li, K., Liu, Y., Ao, X., Chi, J., Feng, J., Yang, H., and He, Q. Reliable representations make a stronger defender: Unsupervised structure refinement for robust gnn. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022.

Li, Q., Wu, X.-M., Liu, H., Zhang, X., and Guan, Z. Label efficient semi-supervised learning via graph filtering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

Li, Y., Jin, W., Xu, H., and Tang, J. Deeprobust: A pytorch library for adversarial attacks and defenses. *arXiv preprint arXiv:2005.06149*, 2020.

Lin, L., Blaser, E., and Wang, H. Graph structural attack by perturbing spectral distance. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022.

Liu, S., Ying, R., Dong, H., Lin, L., Chen, J., and Wu, D. How powerful is implicit denoising in graph neural networks. *arXiv preprint arXiv:2209.14514*, 2022.

Liu, X., Ding, J., Jin, W., Xu, H., Ma, Y., Liu, Z., and Tang, J. Graph neural networks with adaptive residual. In *Advances in Neural Information Processing Systems*, 2021a.

Liu, X., Jin, W., Ma, Y., Li, Y., Liu, H., Wang, Y., Yan, M., and Tang, J. Elastic graph neural networks. In *International Conference on Machine Learning*, 2021b.

Ma, Y., Liu, X., Zhao, T., Liu, Y., Tang, J., and Shah, N. A unified view on graph neural networks as graph signal denoising. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

Moré, J. J. and Sorensen, D. C. Newton's method. Technical report, Argonne National Lab., IL (USA), 1982.

Mujkanovic, F., Geisler, S., Günnemann, S., and Bojchevski, A. Are defenses for graph neural networks robust? In *Advances in Neural Information Processing Systems*, 2022.

Nt, H. and Maehara, T. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550*, 2019.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, 2019.

Pei, H., Wei, B., Chang, K. C.-C., Lei, Y., and Yang, B. Geom-gcn: Geometric graph convolutional networks. In *International Conference on Learning Representations*, 2020.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. Collective classification in network data. *AI Magazine*, 2008.

Shchur, O., Mumme, M., Bojchevski, A., and Günnemann, S. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

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

Vershynin, R. Introduction to the non-asymptotic analysis of random matrices. *arXiv preprint arXiv:1011.3027*, 2010.

Wang, Q., Wang, Y., Zhu, H., and Wang, Y. Improving out-of-distribution generalization by adversarial training with structured priors. In *Advances in Neural Information Processing Systems*, 2022.

Wang, Y., Wang, Y., Yang, J., and Lin, Z. Dissecting the diffusion process in linear graph convolutional networks. In *Advances in Neural Information Processing Systems*, 2021.

Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. Simplifying graph convolutional networks. In *International Conference on Machine Learning*, 2019a.

Wu, H., Wang, C., Tyshetskiy, Y., Docherty, A., Lu, K., and Zhu, L. Adversarial examples for graph data: Deep insights into attack and defense. In *Proceedings of the*

*Twenty-Eighth International Joint Conference on Artificial Intelligence*, 2019b.

Xie, B., Chang, H., Zhang, Z., Wang, X., Wang, D., Zhang, Z., Ying, R., and Zhu, W. Adversarially robust neural architecture search for graph neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

Yang, Y., Liu, T., Wang, Y., Zhou, J., Gan, Q., Wei, Z., Zhang, Z., Huang, Z., and Wipf, D. Graph neural networks inspired by classical iterative algorithms. In *International Conference on Machine Learning*, 2021.

Yang, Z., Cohen, W., and Salakhudinov, R. Revisiting semi-supervised learning with graph embeddings. In *International Conference on Machine Learning*, 2016.

Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., and Leskovec, J. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018.

Zhang, S., Liu, Y., Sun, Y., and Shah, N. Graph-less neural networks: Teaching old MLPs new tricks via distillation. In *International Conference on Learning Representations*, 2022.

Zhang, X. and Zitnik, M. Gnnguard: Defending graph neural networks against adversarial attacks. In *Advances in Neural Information Processing Systems*, 2020.

Zhao, J., Dong, Y., Ding, M., Kharlamov, E., and Tang, J. Adaptive diffusion in graph neural networks. In *Advances in Neural Information Processing Systems*, 2021.

Zhao, K., Kang, Q., Song, Y., She, R., Wang, S., and Tay, W. P. Adversarial robustness in graph neural networks: A hamiltonian approach. In *Advances in Neural Information Processing Systems*, 2023.

Zhao, L. and Akoglu, L. Pairnorm: Tackling oversmoothing in gnns. In *International Conference on Learning Representations*, 2019.

Zhou, B., Li, R., Zheng, X., Wang, Y. G., and Gao, J. Graph denoising with framelet regularizer. *arXiv preprint arXiv:2111.03264*, 2021.

Zhu, D., Zhang, Z., Cui, P., and Zhu, W. Robust graph convolutional networks against adversarial attacks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019.

Zhu, H. and Koniusz, P. Simple spectral graph convolution. In *International Conference on Learning Representations*, 2021.

Zügner, D. and Günnemann, S. Adversarial attacks on graph neural networks via meta learning. In *International Conference on Learning Representations*, 2019a.

Zügner, D. and Günnemann, S. Certifiable robustness and robust training for graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019b.

Zügner, D., Akbarnejad, A., and Günnemann, S. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018.

## A. The Row Summation of the Graph Diffusion Convolution Matrix

We provide the derivations of the row sum of $\mathbf{S} = \frac{1}{\lambda+1}\sum_{k=0}^{K}\left[\frac{\lambda}{\lambda+1}\widetilde{\boldsymbol{\mathcal{A}}}\right]^k$ in this section. Before we derive the row summation of $\mathbf{S}$, we first derive the row summation of $\widetilde{\boldsymbol{\mathcal{A}}}^k$. Note we consider $\widetilde{\boldsymbol{\mathcal{A}}} = \widetilde{\mathbf{D}}^{-1}\widetilde{\mathbf{A}}$ for the simplicity of the proof.

**Lemma 1.** *Consider a probability matrix $\mathbf{P} \in \mathbb{R}^{n\times n}$, where $\mathbf{P}_{ij} \geq 0$. Besides, for all $i$, we have $\sum_{j=1}^{n}\mathbf{P}_{ij} = 1$. Then for any $k \in \mathbb{Z}_+$, we have $\sum_{j=1}^{n}\mathbf{P}_{ij}^k = 1$,*

*Proof.* We give a proof by induction on $k$.
**Base case:** When $k = 1$, the case is true.
**Inductive step:** Assume the induction hypothesis that for a particular $k$, the single case n = k holds, meaning $\mathbf{P}^k$ is true:

$$\forall i, \sum_{j=1}^{n}\mathbf{P}_{ij}^k = 1.$$

As $\mathbf{P}^{k+1} = \mathbf{P}^k\mathbf{P}$, so we have

$$\begin{aligned}
\sum_{j=1}^{n}\mathbf{P}_{ij}^{k+1} &= \sum_{j=1}^{n}\sum_{k=1}^{n}\mathbf{P}_{ik}^k\mathbf{P}_{kj} \\
&= \sum_{k=1}^{n}\sum_{j=1}^{n}\mathbf{P}_{ik}^k\mathbf{P}_{kj} \\
&= \sum_{k=1}^{n}\mathbf{P}_{ik}^k\left(\sum_{j=1}^{n}\mathbf{P}_{kj}\right) \\
&= \sum_{k=1}^{n}\mathbf{P}_{ik}^k = 1,
\end{aligned}$$

which finishes the proof. □

Lemma 1 describes the row summation of $\widetilde{\boldsymbol{\mathcal{A}}}^k$ is 1. Now we can obtain the row summation for $\mathbf{S}$.

Then for any $i$, we have

$$\begin{aligned}
\sum_{j=1}^{n}[\mathbf{S}]_{ij} &= \frac{1}{\lambda+1}\sum_{k=0}^{K}\left(\frac{\lambda}{\lambda+1}\left[\widetilde{\boldsymbol{\mathcal{A}}}\right]_{ij}\right)^k \\
&= \frac{1}{\lambda+1}\sum_{k=0}^{K}\left(\frac{\lambda}{\lambda+1}\right)^k \\
&= 1 - \left(\frac{\lambda}{\lambda+1}\right)^{K+1}.
\end{aligned} \tag{25}$$

## B. Proof of Theorem 1

We first introduce the General Hoeffding Inequality (Hoeffding, 1994), which is essential for bounding $\|\mathbf{S}\boldsymbol{\Upsilon}\|_F^2$.

**Lemma 2. (General Hoeffding Inequality (Hoeffding, 1994))** *Suppose that the variables $X_1, \cdots, X_n$ are independent, and $X_i$ has mean $\mu_i$ and sub-Gaussian parameter $\sigma_i$. Then for all $t \geq 0$, we have*

$$\mathbb{P}\left[\sum_{i=1}^{n}(X_i - \mu_i) \geq t\right] \leq \exp\left\{-\frac{t^2}{2\sum_{i=1}^{n}\sigma_i^2}\right\}. \tag{26}$$

Now, we prove Theorem 1.

*Proof of Theorem 1.* For any entry $[\mathbf{S}\boldsymbol{\Upsilon}]_{ij} = \sum_{p=1}^{n} (\mathbf{S})_{ip} \boldsymbol{\Upsilon}_{pj}$, where $\boldsymbol{\Upsilon}_{pj}$ is a sub-Gaussian variable with parameter $\sigma^2$. By the General Hoeffding inequality 2, we have

$$\begin{aligned} &\mathbb{P}\left(\left|\left[\frac{1}{\lambda+1}\sum_{k=0}^{K}\left(\frac{\lambda}{\lambda+1}\widetilde{\boldsymbol{\mathcal{A}}}\right)^{k}\boldsymbol{\Upsilon}\right]_{ij}\right| \geq t\right) \\ &\leq 2\exp\left\{-\frac{nt^2}{2\tau\left(1-\left(\frac{\lambda}{\lambda+1}\right)^{K+1}\right)^{2}\sigma^2}\right\}, \end{aligned} \tag{27}$$

where $\tau = \max_i \tau_i$ and $\tau_i = n\sum_{j=1}^{n}[\mathbf{S}]_{ij}^2 \Big/ \left(1-\left(\frac{\lambda}{\lambda+1}\right)^{K+1}\right)^{2}$.

Applying union bound (Vershynin, 2010) to all possible pairs of $i \in [n]$, $j \in [n]$, we get

$$\begin{aligned} \mathbb{P}\left(\|\mathbf{S}\boldsymbol{\Upsilon}\|_{\infty,\infty} \geq t\right) &\leq \sum_{i,j}\mathbb{P}\left([\mathbf{S}\boldsymbol{\Upsilon}]_{ij} \geq t\right) \\ &\leq 2n^2\exp\left\{-\frac{nt^2}{2\tau\left(1-\left(\frac{\lambda}{\lambda+1}\right)^{K+1}\right)^{2}\sigma^2}\right\}. \end{aligned} \tag{28}$$

Applying union bound again, we have

$$\begin{aligned} &\mathbb{P}\left(\|\mathbf{S}\boldsymbol{\Upsilon}\|_F^2 \geq t\right) \\ \leq &\sum_{i,j}\mathbb{P}\left(\|\mathbf{S}\boldsymbol{\Upsilon}\|_{\infty,\infty} \geq \sqrt{t}\right) \\ \leq &2n^4\exp\left\{-\frac{nt}{2\tau\left(1-\left(\frac{\lambda}{\lambda+1}\right)^{K+1}\right)^{2}\sigma^2}\right\}. \end{aligned} \tag{29}$$

Choose $t = 2\tau\left(1-\left(\frac{\lambda}{\lambda+1}\right)^{K+1}\right)^{2}(4\log n + \log 2d)/n$ and with probability $1 - 1/d$, we have

$$\|\mathbf{S}\boldsymbol{\Upsilon}\|_F^2 \leq \frac{2\tau\left(1-\left(\frac{\lambda}{\lambda+1}\right)^{K+1}\right)^{2}\sigma^2(4\log n + \log 2d)}{n}, \tag{30}$$

which completes the proof. □

## C. Related Work

**Adversarial Attacks on Graph Structure.** Several defense methods have recently been proposed to counter graph structural attacks (Zhu et al., 2019; Wang et al., 2022; Chang et al., 2022; Xie et al., 2023; Gosch et al., 2023). Wu et al. (2019b) introduce GNN-Jaccard, which works with the graph's adjacency matrix to identify fake edges. RobustGCN (Zügner & Günnemann, 2019b) employs Gaussian distributions in hidden layers to mitigate the effects of attacks. GNN-SVD (Entezari et al., 2020) reduces the rank of the adjacency matrix to counteract NETTACK, which targets high-rank singular components in graph data. Lastly, Pro-GNN (Jin et al., 2020) presents an approach that jointly generates a new structural graph and a robust GNN model from the perturbed graph. By utilizing the additional term in our proposed GADC (IV), we can reconstruct the informative adjacency matrix, thus restoring efficient information flow and defending against adversarial structure attacks.

**Implicit Denoising in GNNs.** Existing graph denoising works primarily focus on graph smoothing techniques (Chen et al., 2014; Wang et al., 2021; Zhou et al., 2021; Chen et al., 2022). It is well-established that GNNs can increase node feature smoothness through neighbor information aggregation, counteracting the influence of noisy features in the GNN's output. Some recent GNN models, such as GLP (Li et al., 2019), $S^2$GC (Zhu & Koniusz, 2021), and IRLS (Yang et al., 2021), are derived from the perspective of signal denoising. Additionally, Ma et al. (Ma et al., 2021) connect signal denoising to popular GNNs by considering the message passing scheme as a process of solving the GSD problem.

## D. Datasets Details

Table 7: Datasets statistics

| **Dataset** | # Nodes | # Edges | # Features | # Classes |
|---|---|---|---|---|
| Cora | 2708 | 5429 | 1433 | 7 |
| Citeseer | 3327 | 4732 | 3703 | 6 |
| Pubmed | 19717 | 44338 | 500 | 3 |
| Cornell | 183 | 295 | 1703 | 5 |
| Texas | 183 | 309 | 1703 | 5 |
| Wisconsin | 251 | 499 | 1703 | 5 |
| Actor | 7600 | 33544 | 931 | 5 |
| Coauthor-CS | 18333 | 81894 | 6805 | 15 |
| Coauthor-Phy | 34493 | 247962 | 8415 | 5 |
| ogbn-products | 2449029 | 61859140 | 100 | 42 |

## E. Hyperparameter Details

We provide details about the hyparatemeters of GADC in Table 8, 9, 10, 11, and 12.

Table 8: The hyper-parameters for GADC (IV) on three citation datasets for defense evaluation against non-adaptive graph structure attacks.

| Model | dataset | runs | lr | epochs | wight decay | hidden | dropout | $K$ | $\lambda$ | perturbation rate |
|---|---|---|---|---|---|---|---|---|---|---|
| GADC (IV) | Cora | 10 | 0.02 | 100 | 1e-5 | 32 | 0.5 | 6 | 1 | 0.25 |
| GADC (IV) | Cora | 10 | 0.02 | 100 | 1e-5 | 32 | 0.5 | 3 | 1 | 0.5 |
| GADC (IV) | Cora | 10 | 0.02 | 100 | 1e-5 | 32 | 0.5 | 1 | 1 | 0.75 |
| GADC (IV) | Citeseer | 10 | 0.02 | 100 | 1e-5 | 32 | 0.5 | 6 | 1 | 0.25 |
| GADC (IV) | Citeseer | 10 | 0.02 | 100 | 1e-5 | 32 | 0.5 | 3 | 1 | 0.5 |
| GADC (IV) | Citeseer | 10 | 0.02 | 100 | 1e-5 | 32 | 0.5 | 1 | 1 | 0.75 |
| GADC (IV) | Pubmed | 10 | 0.02 | 200 | 1e-5 | 32 | 0.5 | 2 | 1 | 0.25 |
| GADC (IV) | Pubmed | 10 | 0.02 | 200 | 1e-5 | 32 | 0.5 | 1 | 1 | 0.5 |
| GADC (IV) | Pubmed | 10 | 0.02 | 200 | 1e-4 | 32 | 0.5 | 1 | 1 | 0.75 |

Table 9: The hyper-parameters for GADC (II) on three citation datasets for denoising evaluation against feature Gaussian noise.

| Model | dataset | runs | lr | epochs | wight decay | hidden | dropout | $K$ | $\lambda$ | $\varepsilon$ |
|---|---|---|---|---|---|---|---|---|---|---|
| GADC (II) | Cora | 100 | 0.2 | 100 | 1e-5 | 0 | 0 | 16 | 32 | 1 |
| GADC (II) | Citeseer | 100 | 0.2 | 100 | 1e-5 | 0 | 0 | 16 | 32 | 1 |
| GADC (II) | Pubmed | 100 | 0.2 | 100 | 1e-5 | 0 | 0 | 16 | 32 | 1 |

Table 10: The hyper-parameters for GADC (II) on two co-author datasets for denoising evaluation against feature Gaussian noise.

| Model | dataset | noise level | runs | lr | epochs | wight decay | hidden | dropout | $K$ | $\lambda$ | $\varepsilon$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| GADC (II) | Coauthor-CS | 0.1 | 10 | 0.2 | 1000 | 1e-7 | 0 | 0 | 16 | 1 | 1 |
| GADC (II) | Coauthor-CS | 1 | 10 | 0.2 | 1000 | 1e-7 | 0 | 0 | 16 | 128 | 1 |
| GADC (II) | Coauthor-Phy | 0.1 | 10 | 0.2 | 1000 | 1e-7 | 0 | 0 | 16 | 1 | 1 |
| GADC (II) | Coauthor-Phy | 1 | 10 | 0.2 | 1000 | 1e-7 | 0 | 0 | 16 | 128 | 1 |

Table 11: The hyper-parameters for GADC (III) on ogbn-products dataset for denoising evaluation against feature Gaussian noise.

| Model | noise level | runs | lr | epochs | hidden | dropout | $K$ | $\lambda$ | $\varepsilon$ | layers | +MLP |
|---|---|---|---|---|---|---|---|---|---|---|---|
| GADC (III) | 0.1 | 10 | 0.01 | 300 | 256 | 0.5 | 128 | 32 | 1e-2 | 3 | True |
| GADC (III) | 1 | 10 | 0.01 | 300 | 256 | 0.5 | 128 | 256 | 1e-2 | 3 | True |

## F. Denoising Performance against Flipping Perturbations

We provide results in Table 13.

## G. Ablation Study on Heterophilic Graphs

In this section, we show the improvement of our proposed GADC (I) on heterophilic graphs with the additional term in the modified transition matrix through a series of ablation studies.

Table 12: The hyper-parameters for GADC (II) on three citation datasets for denoising evaluation against feature flip noise.

| Model | dataset | flip probability | runs | lr | epochs | wight decay | hidden | dropout | $K$ | $\lambda$ | $\varepsilon$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| GADC (II) | Cora | 0.1 | 100 | 0.2 | 100 | 1e-5 | 0 | 0 | 32 | 64 | 1e-5 |
| GADC (II) | Cora | 0.2 | 100 | 0.2 | 100 | 1e-5 | 0 | 0 | 32 | 64 | 1e-5 |
| GADC (II) | Cora | 0.4 | 100 | 0.2 | 100 | 1e-5 | 0 | 0 | 32 | 64 | 1e-1 |
| GADC (II) | Citeseer | 0.1 | 100 | 0.2 | 100 | 1e-5 | 0 | 0 | 32 | 64 | 1e-5 |
| GADC (II) | Citeseer | 0.2 | 100 | 0.2 | 100 | 1e-5 | 0 | 0 | 32 | 64 | 1e-5 |
| GADC (II) | Citeseer | 0.4 | 100 | 0.2 | 100 | 1e-5 | 0 | 0 | 32 | 64 | 1e-5 |
| GADC (II) | Pubmed | 0.1 | 100 | 0.2 | 100 | 1e-5 | 0 | 0 | 32 | 64 | 1e-1 |
| GADC (II) | Pubmed | 0.2 | 100 | 0.2 | 100 | 1e-5 | 0 | 0 | 32 | 64 | 1e-1 |
| GADC (II) | Pubmed | 0.4 | 100 | 0.2 | 100 | 1e-5 | 0 | 0 | 32 | 64 | 1e-1 |

Table 13: Denoising performance over 100 runs against flipping perturbation

| Flipping probability | Cora | | | Citeseer | | | Pubmed | | |
|---|---|---|---|---|---|---|---|---|---|
| | 0.1 | 0.2 | 0.4 | 0.1 | 0.2 | 0.4 | 0.1 | 0.2 | 0.4 |
| MLP | 21.2±7.3 | 21.1±8.0 | 23.3±8.0 | 19.3±3.3 | 18.9±2.8 | 18.9±2.7 | 38.0±6.3 | 39.0±4.7 | 40.6±2.8 |
| GCN | 22.9±13.6 | 19.0±9.4 | 19.0±9.3 | 18.6±3.4 | 18.6±3.1 | 18.5±3.2 | 37.8±6.6 | 38.1±7.1 | 37.6±8.0 |
| GAT | 70.1±1.5 | 65.6±1.5 | 60.0±3.2 | 45.3±2.9 | 39.3±3.4 | 26.0±5.1 | 43.3±2.7 | 49.5±3.2 | 60.0±4.1 |
| APPNP | 75.6±1.2 | 69.8±1.5 | 65.3±1.6 | **56.5±2.2** | 49.1±1.8 | 42.8±3.0 | 43.4±2.7 | 52.3±1.8 | 64.4±1.7 |
| GLP | 32.3±0.8 | 30.8±4.0 | 29.0±6.4 | 19.7±2.7 | 18.9±2.4 | 18.8±2.3 | 42.1±2.0 | 41.5±1.8 | 40.7±0.1 |
| $S^2$GC | 75.0±1.6 | 71.5±2.0 | 63.8±4.4 | 49.9±3.9 | 46.4±3.2 | 43.4±2.9 | 50.4±2.2 | 60.2±1.9 | 69.3±1.6 |
| IRLS | 66.4±2.0 | 61.0±1.9 | 54.7±2.5 | 50.3±2.7 | 45.9±2.1 | 43.8±1.7 | 51.4±4.1 | 60.0±3.8 | 69.0±2.4 |
| GADC (II) | **77.6±1.2** | **75.2±1.4** | **72.8±1.4** | 55.0±3.0 | **51.8±2.4** | **48.7±2.4** | **54.3±2.0** | **63.9±1.6** | **71.6±1.1** |

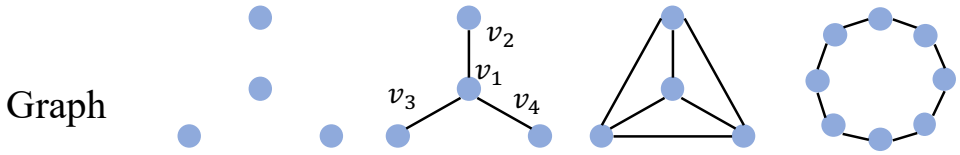
**Datasets.** We use four datasets for fully supervised node classification on heterophilic graphs: Cornell, Texas, Wisconsin, and Actor. For each dataset, we randomly split nodes into 60%, 20%, and 20% for training, validation, and testing, as suggested in Pei et al. (2020).

**Setting and Results.** We evaluate the performance of the additional term on heterophilic graphs by setting $\varepsilon$ to a series of values. We also add a 2-layer MLP with 64 hidden units after the feature aggregation of GADC (I). For GADC (I), we set $\lambda$ to 1 and $K$ to 16. We conduct experiments 100 times and report the mean test accuracy for the node classification task. Note that in each repeated run, we use a different dataset split. From the results summarized in Table 14, it is evident that incorporating the additional term ($\varepsilon \in \{1.0, 2.0\}$) can lead to improved performance compared to the scenario where it is disabled ($\varepsilon = 0$). Furthermore, we notice that when $\varepsilon$ is set to 1.0, the performance improvement is maximized. This observation suggests that these specific perturbation levels could be optimal for the context of our study.

Table 14: Test accuracy with 100 runs of GADC (I) on heterophilic graphs. For these datasets, we use $\varepsilon$ of 0, 1.0, 2.0, 3.0, and 4.0, respectively.

| $\varepsilon$ | 0 | 1.0 | 2.0 | 3.0 | 4.0 |
|---|---|---|---|---|---|
| Cornell | $74.8 \pm 7.0$ | $\mathbf{76.9 \pm 7.6}$ | $76.4 \pm 7.6$ | $69.0 \pm 7.9$ | $57.5 \pm 8.6$ |
| Texas | $74.9 \pm 6.1$ | $\mathbf{77.8 \pm 6.9}$ | $76.7 \pm 7.5$ | $64.6 \pm 7.9$ | $60.8 \pm 8.0$ |
| Wisconsin | $73.2 \pm 5.9$ | $\mathbf{78.2 \pm 6.5}$ | $78.0 \pm 6.7$ | $64.8 \pm 6.5$ | $53.1 \pm 8.2$ |
| Actor | $34.35 \pm 1.35$ | $\mathbf{34.51 \pm 1.41}$ | $25.42 \pm 1.07$ | $25.30 \pm 1.06$ | $25.16 \pm 1.05$ |

## H. Illustration of Higher-order Graph Connectivity Factor on Various Graphs




| Metric | $\mathcal{G}_1$ | $\mathcal{G}_2$ | $\mathcal{G}_3$ | $\mathcal{G}_4$ |
|---|---|---|---|---|
| #Nodes | 4 | 4 | 4 | 8 |
| $\tau$ | 4 | 1.14 | 1 | 1.03 |
| $\tau \log n / n$ | 1.39 | 0.39 | 0.35 | 0.27 |

Figure 2: An illustration of $\tau$ on various graph structures. $\mathcal{G}_1$: nodes are isolated; $\mathcal{G}_2$: a star graph with 4 nodes; $\mathcal{G}_3$: a complete graph with 4 nodes. For computing $\tau$, we set $\lambda$ and $K$ as 32. $\tau$ has a smaller value if the graph has good connectivity.

In this case study, we present four illustrative samples in Figure 2: $G_1$, $G_2$, $G_3$, and $G_4$.

Graphs $G_1$, $G_2$, and $G_3$ have the same number of nodes. However, the nodes in $G_1$ are isolated, $G_2$ consists of a single connected component with a central node $v_1$, and $G_3$ is a complete graph where each node is connected to every other node. Additionally, we include a larger graph, $G_4$, to analyze the influence of graph size on the denoising effect. From Figure 2, we can derive the following insights:

- No Denoising on Isolated Graph ($G_1$): Since the nodes in $G_1$ are isolated, there is no denoising effect, as indicated by the large value of $\tau \log n/n$.
- Best Denoising on Complete Graph ($G_3$): The complete graph $G_3$ exhibits the best denoising effect among the graphs of the same size. This is because the elements in each row are uniformly distributed, leading to the lowest possible value of $\tau$.
- Central Node Impact on Connected Graph ($G_2$): Although $G_2$ has only one connected component, the presence of a central node $v_1$ creates an imbalance in the value of elements in each row. Consequently, $\tau$ tends to be larger compared to $G_3$.

- Denoising on Decentralized Larger Graph ($G_4$): The decentralized structure of $G_4$ also results in a smaller value of $\tau$, indicating a good denoising effect.
- Influence of Graph Size: Larger graphs, such as $G_4$, tend to have a better denoising effect due to their size.

By analyzing these graphs, we get a deeper understanding of how graph structure and size influence the denoising effect.

## I. Reproducibility

We use Pytorch (Paszke et al., 2019) and PyG (Fey & Lenssen, 2019) to implement GADC. All the experiments in this work are conducted on a single NVIDIA Tesla A100 with 80GB memory size. The software that we use for experiments are Python 3.6.8, pytorch 1.9.0, pytorch-scatter 2.0.9, pytorch-sparse 0.6.12, pyg 2.0.3, ogb 1.3.4, numpy 1.19.5, torchvision 0.10.0, and CUDA 11.1.

## J. Work Statement

This is an extended work based on our manuscript (Liu et al., 2022) that appeared in 2022 NeurIPS GFrontiers. Songtao Liu independently extends Liu et al. (2022) and rewrites the initial versions of this work.