A Appendix

A.1 Masking Rate and Robustness

Table 6 shows the performance of GraphMAE at different masking ratios on adversarial graph data.

Table 6: Robustness performance on node classification task of the random masking strategy used in GraphMAE on datasets perturbed by MetaAttack(M) with a 25% perturbation rate, **Mask Ratio** represents different masking ratios

Mask Ratio	M-Cora	M-Citeseer	M-CoraML
20%	49.30%	60.78%	36.03%
40%	48.64%	61.91%	43.02%
60%	49.40%	62.32%	45.86%
80%	52.63%	62.91%	46.25%

A.2 Graph Self-Supervised Learning Graph Self-Supervised Learning (GSSL) can be roughly divided into two categories: contrastive methods and generative methods.

A.2.1 Contrastive Methods The purpose of graph contrastive learning (GCL) is to learn an encoder that maximizes the mutual information between different graph views. GCL has achieved impressive performance in many graph tasks, but its success heavily relies on stable training strategies [2, 29], carefully designed augmented views [3, 5, 6], and high-quality negative samples [7], which inevitably increases the cost of expert experience. DGI [2] follows the InfoMax principle [30] by learning node representations through local-global mutual information maximization. AD-GCL [29] breaks away from the InfoMax principle and is inspired by the Graph Information Bottleneck (GIB) [31]. It optimizes adversarial graph augmentation strategies to avoid capturing redundant information during training. GRACE [5] proposes edge removal and feature masking augmentation, GCA [6] adopts probabilistic adaptive augmentation, while MVGRL [3] employs diffusion graph as augmentation view. Regarding negative samples, ProGCL [7] proposes the use of Expectation Maximization (EM) algorithm [32] to obtain more effective negative samples.

A.2.2 Generative methods Generative methods, which aiming to directly reconstruct input data, greatly reduce the empirical cost compared to contrastive methods. The inspiration can be traced back to the Autoencoder [33] proposed in 2006. In 2016, GAE and VGAE [24] introduce GCN-based encoders and decoders for link prediction, representing a milestone on graph. In 2017, EP [34] proposes a generative framework based on node attribute propagation, while MGAE [35] focuses on recovering raw features from perturbed features. In 2018, ARVGA [36] observes that most methods are non-regularized and often learn degenerate identity mappings, introducing two new adversarial variants to enforce the latent code to match a prior distribution. In 2019, GALA [37] notices limitations in the decoder and designes a decoder based on Laplacian sharpening. After 2020, there are further explorations such as [38, 39], but the development has been relatively slow.

In 2022, inspired by BERT [8] and MAE [9], the introduction of the masking concept ushered generative GSSL into a new era. GraphMAE [1], as a representative example, reconstructs features using masking strategies and designs scaled cosine error. After 2023, Masked graph autoencoders (MGAEs) have become mainstream. GraphMAE2 [10] further extends this idea by designing the strategies of multi-view random remask decoding and latent representation prediction for feature reconstruction, aiming to reduce excessive reliance on feature discriminability. S2GAE [12] proposes direction-aware graph masking and cross-correlation decoder. MaskGAE [11] adopts edge-wise and path-wise random masking, also introduces a degree decoder to alleviate the problem of structural information overfitting. RARE [40] improves the certainty of inferring masked data and the reliability of self-supervised mechanisms by further masking and reconstructing node samples in a high-order latent feature space.

However, these state-of-the-art methods have only demonstrated effectiveness on reliable graph data. When the graph data is unreliable or maliciously attacked, the randomness introduced by their masking strategies becomes a key factor leading to the degradation of model robustness.

A.3 Attacks and Defenses

A.3.1 Gray-box Non-adaptive Attacks Using classic gray-box poisoning attack, MetaAttack [14] as an example, it utilizes a surrogate model for the attack, which can be formulated mathematically as a bilevel optimization problem:

$$\min_{\hat{G}\in\Phi(G)} \mathcal{L}_{atk}(f_{\theta^*}(\hat{G})) \quad s.t. \quad \theta^* = \underset{\theta}{argmin}\, \mathcal{L}_{train}(f_\theta(\hat{G})) \tag{A.1}$$

$\Phi(G)$ represents a set of graphs that satisfies the disturbance budget constraint Δ. Δ indicates a limit on the number of changes $\|A - \hat{A}\| \leq \Delta$. $\mathcal{L}_{atk}$ is the attack loss function, could be $-\mathcal{L}_{train}$ or $-\mathcal{L}_{self}$.

A.3.2 Adaptive Attacks To provide a more comprehensive evaluation of robustness, [41] presents an interesting research point: almost all defenses are evaluated against non-adaptive attacks, leading to overly optimistic robustness estimates. Therefore, they categorize 49 defense methods and select the most representative method from each category to design targeted adaptive attack methods. The adversarial graphs generated by these attacks can be bundled together to test other defenses and can be considered a minimal standard for evaluating the adaptive robustness of defense models. In the unit test, the datasets are centered around Citeseer and Cora-ML. for each representative model, there are 5 random data splits, each containing poisoning and evasion attacks. The attack budget ranges from 0% to 15%, resulting in approximately 2700 testable graphs in total. The unit test module used in this paper is adapted from the following repository: https://github.com/LoadingByte/are-gnn-defenses-robust

A.3.3 Defenses Numerous efforts have already been devoted to studying the robustness of graphs, with much of the focus centered around semi-supervised learning scenarios. From the perspective of purifying graph structures, GLCN [42] learns the connection relationship between two nodes through a single-layer neural network. ProGNN [43] observes that real-world graphs should have low-rank and sparse properties. From the perspective of aggregation, RGCN [44], GNNGuard [26], MedianGCN and TrimmedGCN [45] strive to design robust messaging mechanisms. From the perspective of preprocessing, GCN-Jaccard [27] and GCN-SVD [19] preprocess the perturbation graph, make a priority judgment before training. In an unsupervised setting, contrastive methods STABLE [46], SPMGAE [47], AD-GCL [29], and GASSL [48] ingeniously utilize redundant information for robust training. [16] enhances the attack methods by incorporating certified robustness, while RES-GCL [49] pioneers the exploration of certified robustness on GCL.

A.4 Training Algorithm To facilitate understanding, we refer to the proposed framework as FilterMGAE. We embed it into GraphMAE as an example, with detailed explanations provided in Algorithm 1. By ranking and sampling the data in the preprocessing stage (Lines 1-3), we effectively clean and filter the data without increasing space overhead. This enables the preparation of a more compact yet precise dataset for pretraining state-of-the-art MGAEs with high scalability. Our framework enhances training efficiency, reduces data storage requirements, and supports the development of more robust encoders in these advanced MGAEs.

Algorithm 1 Filtered Masked Graph Autoencoder

Input: A structure-perturbed Graph $\hat{\mathcal{G}} = \{\mathcal{V}, \hat{\mathcal{A}}, \mathcal{X}\}$, GNN-based Encoder $\mathcal{F}_{e_\theta}(\cdot)$ and Decoder $\mathcal{F}_{d_\theta}(\cdot)$, Weighted Ranking Scheme $\omega \in \{\omega_{degree}, \omega_{csim}, \omega_K, \omega_{dis}, \omega_{mix}\}$, Node Re-ranking Scheme $\tau \in \{\tau_{fix}, \tau_{sample}\}$, Median and Interval Hyperparameters M_k, M_m, r_k, r_m for kept and masked reconstruction node boxes;

Output: Learned Robust $\mathcal{F}_e(\cdot)$

1: Generate node weights according to the selected ranking weight scheme ω;
2: Utilize node weights ω and re-rank the nodes using τ to obtain the ranking sequence N_{ranked};
3: Use a box-based sampling strategy to sample N_{keep} and N_{Masked} from N_{ranked};
4: **while** *not converged* **do**
5: $\quad \mathcal{Z}_{keep} = \mathcal{F}_{e_\theta}(\hat{\mathcal{A}}, \mathcal{X}_{N_{keep}})$;
6: $\quad \mathcal{X}_{N_{Masked}} = \mathcal{F}_{d_\theta}(\varphi(\hat{\mathcal{A}}), \phi(\mathcal{Z}_{keep}))$;
7: $\quad$ Calculate the reconstruction loss in the same manner as described in [1];
8: $\quad$ Update e_θ and d_θ by gradient descent;
9: **end while**
10: **return** $\mathcal{F}_e(\cdot)$;
11: $\mathcal{F}_e(\cdot)$ generates node embeddings on $\hat{\mathcal{G}}$;
12: Predict node labels on downstream tasks based on the embeddings;
13: Return predicted labels;

A.5 Datasets and Attacks Settings Following the settings in *Nettack* [50], we only consider the largest connected component (LCC). Since Polblogs does not have features, we set the features of Polblogs to be an $n \times n$ identity matrix following [43, 46]. The details of the datasets are presented in Table 7.

Table 7: Details of the largest connected component(LCC) for each dataset

Datasets	N_{LCC}	E_{LCC}	Features	Classes
Citeseer	2110	3668	3703	6
Cora	2485	5069	1433	7
Cora-ML	2810	7981	2879	7
Polblogs	1222	16714	/	2
Pubmed	19717	44324	500	3
ogbn-arxiv	169343	1157799	128	40

The code implementation for attacking the graph using different attack methods can be found at the

following links.

- **Metattack**: MetaAttack
- **Heuristic Attack**: Heuristic Attack
- **DICE**: DICE
- **Random**: Random

A.6 Baselines on Attacked Graph Data The code implementation for all baselines can be found at the following links.

- **GAE/VGAE**: GAE/VGAE
- **S2GAE**: S2GAE
- **MaskGAE**: MaskGAE
- **GraphMAE**: GraphMAE
- **GraphMAE2**: GraphMAE2

We fine-tuned the baseline parameters for each attacked dataset to achieve optimal robustness performance. The parameter settings for the baselines on each attacked dataset are as follows:

- **GAE**: we conduct experiments using different *out_channels* = [16, 32, 64] on each attacked dataset and recorded the best result obtained.
- **VGAE**: we conduct experiments using different *out_channels* = [16, 32, 64, 128, 256, 512, 1024, 2048] on each attacked dataset and recorded the best result obtained.
- **S2GAE**: we conduct experiments using different *hidden* = [64, 128, 256] on each attacked dataset and recorded the best result obtained.
- **MaskGAE**: we conduct experiments using different *encoder_channels* = [64, 128, 256], Mask = edge, path on each attacked dataset and recorded the best result obtained.
- **GraphMAE**: we conduct experiments using different *encoder_channels* = [64,256,512,1024], Mask_ratio = 0.2, 0.4, 0.5, 0.6, 0.8 on each attacked dataset and recorded the best result obtained.
- **GraphMAE2**: we conduct experiments using different *encoder_channels* = [64,256,512,1024], Mask Strategy = random, fixed, Mask_ratio = 0.2, 0.4, 0.5, 0.6, 0.8 on each attacked dataset and recorded the best result obtained.

A.7 Evaluation on Reliable Graph Data (RQ5) To verify whether our method sacrifices performance on reliable graph data due to robustness considerations, we conduct broader comparative experiments. We comprehensively compare our method with supervised learning models (**GCN** [51], **GAT** [52]), contrastive SSL models (**DGI** [2], **MVGRL** [3], **BGRL** [4], **GRACE** [5]), and generative SSL models mentioned in the previous section.

Table 8: Performance of different types of methods on node classification datasets.

	Cora	**Citeseer**	**Pubmed**
Supervised			
GCN	$81.60_{\pm 0.14}$	$70.90_{\pm 0.14}$	$79.47_{\pm 0.12}$
GAT	$83.00_{\pm 0.70}$	$72.50_{\pm 0.70}$	$79.00_{\pm 0.30}$
Contrastive SSL			
DGI	$82.30_{\pm 0.60}$	$71.80_{\pm 0.70}$	$76.80_{\pm 0.60}$
MVGRL	$83.50_{\pm 0.05}$	$73.30_{\pm 0.50}$	$80.10_{\pm 0.70}$
BGRL	$82.40_{\pm 0.50}$	$70.80_{\pm 0.80}$	$80.70_{\pm 0.50}$
GRACE	$80.50_{\pm 0.80}$	$70.90_{\pm 0.50}$	$80.60_{\pm 0.50}$
Generative SSL			
GAE	$77.04_{\pm 0.63}$	$64.71_{\pm 0.59}$	$78.26_{\pm 1.22}$
VGAE	$78.00_{\pm 0.55}$	$64.96_{\pm 0.69}$	$78.61_{\pm 0.94}$
MaskGAE	$84.05_{\pm 0.18}$	$73.49_{\pm 0.59}$	$\mathbf{83.06_{\pm 0.22}}$
S2GAE[1]	$81.50_{\pm 0.60}$	$67.40_{\pm 0.60}$	$79.00_{\pm 0.25}$
GraphMAE	$84.20_{\pm 0.40}$	$73.40_{\pm 0.40}$	$81.10_{\pm 0.40}$
GraphMAE2	$84.50_{\pm 0.60}$	$73.40_{\pm 0.30}$	$81.40_{\pm 0.50}$
Ours	$\mathbf{84.60_{\pm 0.30}}$	$\mathbf{73.50_{\pm 0.40}}$	$81.00_{\pm 0.25}$

* The results are from the corresponding papers. If not mentioned, we use the author's official code to implement it and tune the parameters for the dataset.
[1] The results of S2GAE differ from those in the paper because it evaluates the classification accuracy of all nodes, whereas we evaluate only the test nodes.

Based on the experimental results in Table 8, it can be observed that our method does not sacrifice much performance on reliable graphs due to considering robustness. It can perform comparably to state-of-the-art methods. Additionally, we find that introducing more randomness is meaningful on reliable graph data. In our approach, the information brought by small-scale box data is limited. Therefore, when we increase r_k and r_m to cover the entire graph as much as possible and choose weighted sampling scheme, it will produce satisfactory results. In this scenario, our method effectively simulates advanced approaches designed for reliable graphs, which aligns with our previous analysis.

A.8 Evaluation on Robust Variants (RQ6) We also integrate mainstream defense methods to design other robust variants for GraphMAE. We preprocess the graph using methods based on Jaccard [27] and SVD [19]. In Jaccard, we eliminate edges below a threshold τ by computing the Jaccard similarity of connected

node features. In SVD, we purify the perturbed graph using low-rank approximation. The preprocessed graph is then used in place of the original graph for the pretraining process, and downstream task performance is evaluated on the original graph. They are denoted as Variant **G-JA** and Variant **G-SVD**, respectively. We also replace the encoder of GraphMAE with defense model GNNGuard [26] throughout the pretraining and evaluation process, denoted as Variant **G-Guard**. We compare these robustness variants against our method under three different attack ratios of MetaAttack.

Table 9: Performance of our method and robust variants on Cora and Cora-ML under MetaAttack.

Dataset	Ptb Rate	G-JA	G-SVD	G-Guard	Ours
Cora	0(%)	84.80(%)	78.99(%)	82.44(%)	**85.76(%)**
	15(%)	70.57(%)	63.23(%)	64.24(%)	**72.23(%)**
	25(%)	57.65(%)	50.20(%)	51.13(%)	**62.12(%)**
Cora-ML	0(%)	85.99(%)	70.06(%)	85.05(%)	**86.92(%)**
	15(%)	76.97(%)	46.37(%)	66.73(%)	**78.91(%)**
	25(%)	62.59(%)	38.61(%)	43.95(%)	**64.89(%)**

According to the experimental results in Table 9, we find that when using different robust tools to insert into GraphMAE, our method achieves better robustness performance in a more lightweight and efficient manner.

A.9 Hyperparameter Analysis We utilize distribution-based ranking weights and a fixed re-ranking scheme on Cora under MetaAttack with a 25% perturbation ratio, as this strategy yields the highest robustness performance.

- To explore M_k, we fixed $M_m = 0.8$, $r_m = 0.2$, $r_k = 0.05$ and let M_k vary from 0.05 to 0.55 in intervals of 0.1.
- To explore r_k, we fixed $M_m = 0.9$, $r_m = 0.1$, $M_k = 0.4$ and let r_k vary from 0.1 to 0.4 in intervals of 0.1.
- To explore M_m, we fixed $M_k = 0.05$, $r_k = 0.05$, $r_m = 0.05$ and let M_m vary from 0.95 to 0.15 in intervals of 0.2.
- To explore r_m, we fixed $M_k = 0.05$, $r_k = 0.05$, $M_m = 0.55$ and let r_m vary from 0.05 to 0.45 in intervals of 0.1.