Are Defenses for Graph Neural Networks Robust?

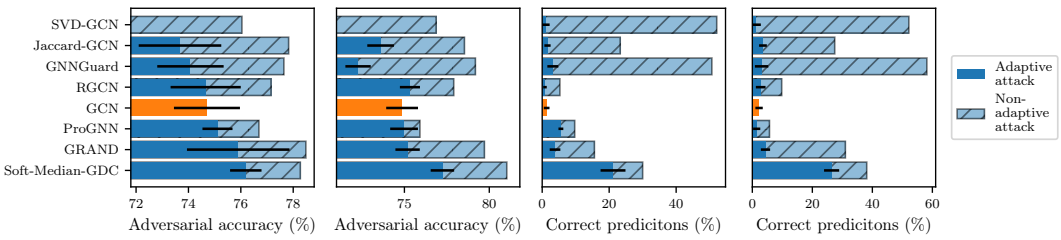
Felix Mujkanovic[1]*, **Simon Geisler**[1]*, **Stephan Günnemann**[1], **Aleksandar Bojchevski**[2]
[1]Dept. of Computer Science & Munich Data Science Institute, Technical University of Munich
[2]CISPA Helmholtz Center for Information Security
{f.mujkanovic, s.geisler, s.guennemann}@tum.de | bojchevski@cispa.de

Abstract

A cursory reading of the literature suggests that we have made a lot of progress in designing effective adversarial defenses for Graph Neural Networks (GNNs). Yet, the standard methodology has a serious flaw – virtually all of the defenses are evaluated against non-adaptive attacks leading to overly optimistic robustness estimates. We perform a thorough robustness analysis of 7 of the most popular defenses spanning the entire spectrum of strategies, i.e., aimed at improving the graph, the architecture, or the training. The results are sobering – most defenses show no or only marginal improvement compared to an undefended baseline. We advocate using custom adaptive attacks as a gold standard and we outline the lessons we learned from successfully designing such attacks. Moreover, our diverse collection of perturbed graphs forms a (black-box) unit test offering a first glance at a model's robustness.[1]

1 Introduction

The vision community learned a bitter lesson – we need specific carefully crafted attacks to properly evaluate the adversarial robustness of a defense. Consequently, adaptive attacks are considered the gold standard [44]. This was not always the case; until recently, most defenses were tested only against relatively weak static attacks. The turning point was Carlini & Wagner [3]'s work showing that 10 methods for detecting adversarial attacks can be easily circumvented. Shortly after, Athalye et al. [1] showed that 7 out of the 9 defenses they studied can be broken since they (implicitly) rely on obfuscated gradients. So far, this bitter lesson is completely ignored in the graph domain.



(a) Global, Poisoning (b) Global, Evasion (c) Local, Poisoning (d) Local, Evasion

Figure 1: Adaptive attacks draw a different picture of robustness. All defenses are less robust than reported, with an undefended GCN [33] outperforming some. We show results on Cora ML for both poisoning (attack before training) and evasion (attack after training), and both global (attack the test set jointly) and local (attack individual nodes) setting. The perturbation budget is relative w.r.t. the #edges for global attacks (5% evasion, 2.5% poisoning) and w.r.t. the degree for local attacks (100%). In (a)/(b) SVD-GCN is catastrophically broken – our adaptive attacks reach 24%/9% (not visible). Note that our non-adaptive attacks are already stronger than what is typically used (see § 5).

*equal contribution [1] Project page: https://www.cs.cit.tum.de/daml/are-gnn-defenses-robust/

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

Virtually no existing work that proposes an allegedly robust Graph Neural Network (GNN) evaluates against adaptive attacks, leading to overly optimistic robustness estimates. To show the seriousness of this methodological flaw we categorize 49 works that propose a robust GNN and are published at major conferences/journals. We then choose one defense per category (usually the most highly cited). Not surprisingly, we show that none of the assessed models are as robust as originally advertised in their respective papers. In Fig. 1 we summarize the results for 7 of the most popular defenses, spanning the entire spectrum of strategies (i.e., aimed at improving the graph, the architecture, or the training, see Table 1).

We see that in both local and global settings, as well as for both evasion and poisoning, the adversarial accuracy under our adaptive attacks is significantly smaller compared to the routinely used non-adaptive attacks. Even more troubling is that many of the defenses perform worse than an undefended baseline (a vanilla GCN [33]). Importantly, the 7 defenses are not cherry-picked. We report the results for each defense we assessed and selected each defence before running any experiments.

Adversarial robustness measures the local generalization capabilities of a model, i.e., sensitivity to (bounded) worst-case perturbations. Certificates typically provide a lower bound on the actual robustness while attacks provide an upper bound. Since stronger attacks directly translate into tighter bounds our goal is to design the strongest attack possible. Our adaptive attacks have perfect knowledge of the model, the parameters, and the data, including all defensive measures. In contrast, non-adaptive attacks (e.g., transferred from an undefended proxy or an attack lacking knowledge about defense measures) only show how good the defense is at suppressing a narrow subset of input perturbations.[2]

Tramer et al. [44] showed that even adaptive attacks can be tricky to design with many subtle challenges. The graph domain comes with additional challenges since graphs are typically sparse and discrete and the representation of any node depends on its neighborhood. For this reason, we describe the recurring themes, the lessons learned, and our systematic methodology for designing strong adaptive attacks for all examined models. Additionally, we find that defenses are *sometimes* sensitive to a common attack vector and transferring attacks can also be successful. Thus, the diverse collection of perturbed adjacency matrices resulting from our attacks forms a (black-box) unit test that any truly robust model should pass before moving on to adaptive evaluation. In summary:

- We survey and categorize *49 defenses* published across prestigious machine learning venues.
- We design custom attacks for 7 defenses (14%), covering the spectrum of defense techniques. All examined models forfeit a large fraction of previously reported robustness gains.
- We provide a transparent methodology and guidelines for designing strong adaptive attacks.
- Our collection of perturbed graphs can serve as a robustness unit test for GNNs.

2 Background and preliminaries

We follow the most common setup and assume GNN [20, 33] classifiers $f_\theta(\mathbf{A}, \mathbf{X})$ that operate on a symmetric binary adjacency matrix $\mathbf{A} \in \{0,1\}^{n\times n}$ with binary node features $\mathbf{X} \in \{0,1\}^{n\times d}$ and node labels $\mathbf{y} \in \{1, 2, \ldots, C\}^n$ where C is the number of classes, n is the number of nodes, and m the number of edges. A poisoning attack perturbs the graph (flips edges) prior to training, optimizing

$$\max_{\tilde{\mathbf{A}}\in\Phi(\mathbf{A})} \ell_{\text{attack}}(f_{\theta^*}(\tilde{\mathbf{A}}, \mathbf{X}), \mathbf{y}) \quad \text{s.t.} \quad \theta^* = \arg\min_\theta \ell_{\text{train}}(f_\theta(\tilde{\mathbf{A}}, \mathbf{X}), \mathbf{y}) \tag{1}$$

where ℓ_{attack} is the attacker's loss, which is possibly different from ℓ_{train} (see § 4). In an evasion attack, θ^* is kept fixed and obtained by training on the clean graph $\min_\theta \ell_{\text{train}}(f_\theta(\mathbf{A}, \mathbf{X}), \mathbf{y})$. In both cases, the locality constraint $\Phi(\mathbf{A})$ enforces a budget Δ by limiting the perturbation to an L_0-ball around the clean adjacency matrix: $\|\tilde{\mathbf{A}} - \mathbf{A}\|_0 \leq 2\Delta$. Attacks on $\mathbf{X}$ also exist, however, this scenario is not considered by the vast majority of defenses. For example, only one out of the seven examined ones also discusses feature perturbations. We refer to § D for more details on adaptive feature attacks.

Threat model. Our attacks aim to either cause misclassification of the entire test set (*global*) or a single node (*local*). To obtain the strongest attack possible (i.e., tightest robustness upper bound), we use white-box attacks. We do not constrain the attacker beyond a simple budget constraint that enforces a maximum number of perturbed edges. For our considerations on unnoticeability, see § A.

[2] From a security perspective non-adaptive attacks (typically transfer attacks) are also relevant since a real-world adversary is unlikely to know everything about the model and the data.

Greedy attacks. Attacking a GNN typically corresponds to solving a constrained discrete non-convex optimization problem that – evident by this work – is hard to solve. Commonly, approximate algorithms are used to to tackle these optimization problems. For example, the single-step Fast Gradient Attack (FGA) flips the edges whose gradient (i.e., $\nabla_{\mathbf{A}} \ell_{\text{train}}(f_{\theta^*}(\mathbf{A}, \mathbf{X}), \mathbf{y})$) most strongly indicates so. On the other hand, Nettack [67] and Metattack [66] are greedy multi-step attacks. The greedy approaches have the nice side-effect that an attack for a high budget Δ directly gives all attacks for budgets lower than Δ. On the other hand, they tend to be relatively weaker.

Projected Gradient Descent (PGD). Alternatively, PGD [53] has been applied to GNNs where the discrete adjacency matrix is relaxed to $[0, 1]^{n \times n}$ during the gradient-based optimization and the resulting weighted change reflects the probability of flipping an edge. After each gradient update, the changes are projected back such that the budget holds in expectation $\|\mathbb{E}[\tilde{\mathbf{A}}] - \mathbf{A}\|_0 \leq 2\Delta$. Finally, multiple samples are obtained and the strongest perturbation $\tilde{\mathbf{A}}$ is chosen that obeys the budget Δ. The biggest caveats while applying L_0-PGD are the relaxation gap and limited scalability (see Geisler et al. [17] for a detailed discussion and a scalable alternative).

Evasion vs. poisoning. Evasion can be considered the easier setting from an attack perspective since the model is fixed f_{θ^*}. For poisoning, on the other hand, the adjacency matrix is perturbed before training (Eq. 1). Two general strategies exist for poisoning attacks: (1) transfer a perturbed adjacency matrix from an evasion attack [67]; or (2) attack directly by, e.g., unrolling the training procedure to obtain gradients through training [66]. Xu et al. [53] propose to solve Eq. 1 with alternating optimization which was shown to be even weaker than the evasion transfer (1). Note that evasion is particularly of interest for inductive learning and poisoning for transductive learning.

3 Adversarial defenses

We select the defenses s.t. we capture the entire spectrum of methods improving robustness against structure perturbations. For the selection, we extend the taxonomy proposed in [21]. We selected the subset without cherry-picking based on the criteria elaborated below before experimentation.

Taxonomy. The top-level categories are *improving the graph* (e.g., preprocessing), *improving the training* (e.g., adversarial training or augmentations), and *improving the architecture*. Many defenses for structure perturbations either fall into the category of improving the graph or adaptively weighting down edges through an improved architecture. Thus, we introduce further subcategories. Similar to [21]'s discussion, unsupervised improvement of the graph finds clues in the node features and graph structure, while supervised improvement incorporates gradient information from the learning objective. Conversely, for adaptive edge weighting, we identify three prevalent approaches: rule-based (e.g., using a simple metric), probabilistic (e.g., modeling a latent distribution), and robust aggregations (e.g., with guarantees). We assign each defense to the most fitting taxon (details in § B).

Selected defenses. To evaluate a diverse set of defenses, we select one per leaf taxon.[3] We prioritize highly cited defenses published at renowned venues with publicly available code. We implement all defenses in one unified pipeline. We present the categorization of defenses and our selection in Table 1. Similarly to Tramer et al. [44], we exclude defenses in the "robust training" category (see § C for a discussion). Two of the three models in the "miscellaneous" category report some improvement in robustness, but they are not explicitly designed for defense purposes so we exclude them from our study. Some works evaluate only against evasion [48], others only poisoning [12, 15, 58], and the rest tackle both [17, 30, 63]. In some cases the evaluation setting is not explicitly stated and inferred by us. For completeness, we consider each defense in all four settings (local/global and evasion/poisoning). Next, we provide a short summary of the key ideas behind each defense (details in § E).

Improving the graph. The feature-based *Jaccard-GCN* [48] uses a preprocessing step to remove all edges between nodes whose features exhibit a Jaccard similarity below a certain threshold. This was motivated by the homophily assumption which is violated by prior attacks that tend to insert edges between dissimilar nodes. The structure-based *SVD-GCN* [12] replaces the adjacency matrix with a low-rank approximation prior to plugging it into a regular GNN. This defense was motivated by the observation that the perturbations from Nettack tend to disproportionately affect the high-frequency spectrum of the adjacency matrix. The key idea in *ProGNN* [30] is to learn the graph structure by

[3] The only exception is unsupervised graph improvement, as it contains two of the most popular approaches, which rely on orthogonal principles. One filters edges based on the node features [48], the other uses a low-rank approximation of the adjacency matrix [12].

Table 1: Categorization of selected defenses. Our taxonomy extends the one by Günnemann [21].

Taxonomy			Selected Defenses	Other Defenses
Improving graph	Unsupervised		Jaccard-GCN [48] SVD-GCN [12]	[10, 26, 50, 59, 60]
	Supervised		ProGNN [30]	[51, 43, 56]
Improving training	Robust training		n/a (see § C)	[6, 9, 14, 22, 27, 28, 41, 52, 53, 54]
	Further training principles		GRAND [15]	[5, 11, 29, 39, 42, 55, 61, 64, 65]
Improving architecture	Adaptively weighting edges	Rule-based	GNNGuard [58]	[31, 36, 37, 57]
		Probabilistic	RGCN [63]	[8, 13, 24, 25, 38]
		Robust agg.	Soft-Median-GDC [17]	[7, 16, 47]
	Miscellaneous		n/a (see above)	[40, 46, 49]

alternatingly optimizing the parameters of the GNN and the adjacency matrix (the edge weights). The loss for the latter includes the standard cross-entropy loss, the distance to the original graph, and three other objectives designed to promote sparsity, low rank, and feature smoothness.

Improving the training. *GRAND* [15] relies on random feature augmentations (zeroing features) coupled with neighbourhood augmentations $\bar{\mathbf{X}} = (\mathbf{AX} + \mathbf{AAX} + \cdots)$. All randomly augmented copies of $\bar{\mathbf{X}}$ are passed through the same MLP that is trained with a consistency regularization loss.

Improving the architecture. *GNNGuard* [58] filters edges in each message passing aggregation via cosine-similarity (smoothed over layers). In the first layer of *RGCN* [63] we learn a Gaussian distribution over the feature matrix and the subsequent layers then manipulate this distribution (instead of using point estimates). For the loss we then sample from the resulting distribution. In addition, in each layer, RGCN assigns higher/lower weights to features with low/high variance. *Soft-Median-GDC* [17] replaces the message passing aggregation function in GNNs (typically a weighted mean) with a more robust alternative by relaxing the median using differentiable sorting.

Common themes. One theme shared by some defenses is to first discover some property that can discriminate clean from adversarial edges (e.g., high vs. low feature similarity), and then propose a strategy based on that property (e.g., filter low similarity edges). Often they analyze the edges from only a single attack such as Nettack [67]. The obvious pitfall of this strategy is that the attacker can easily adapt by restricting the adversarial search space to edges that will bypass the defense's (implicit) filter. Another theme is to add additional loss terms to promote some robustness objectives. Similarly, the attacker can incorporate the same terms in the attack loss to negate their influence.

4 Methodology: How to design strong adaptive attacks

In this section, we describe our general methodology and the lessons we learned while designing adaptive attacks. We hope these guidelines can serve as a reference for testing new defenses.

Step 1 – Understand how the defense works and categorize it. For example, some defenses rely on preprocessing which filters out edges that meet certain criteria (e.g., Jaccard-GCN [48]). Others introduce additional losses during training (e.g., GRAND [15]) or change the architecture (e.g., RGCN [63]). Different defenses might need different attacks or impose extra requirements on them.

Step 2 – Probe for obvious weaknesses. Some examples include: (a) transfer adversarial edges from another (closely related) model (see also § 6); (b) use a gradient-free (black-box) attack. For example, in our local experiments, we use a *Greedy Brute Force* attack: in each step, it considers all possible single edge flips and chooses the one that contributes most to the attack objective (details in § A).

Step 3 – Launch a gradient-based adaptive attack. For rapid prototyping, use a comparably cheap attack such as FGA, and later advance to stronger attacks like PGD. For poisoning, strongly consider meta-gradient-based attacks like Metattack [66] that unroll the training procedure, as they almost always outperform just transferring perturbations from evasion. Unsurprisingly, we find that applying PGD [53] on the meta gradients often yields even stronger attacks than the greedy Metattack, and we refer to this new attack as *Meta-PGD* (details in § A).

Step 4 – Address gradient issues. Some defenses contain components that are non-differentiable, lead to exploding or vanishing gradients, or obfuscate the gradients [1]. To circumvent these issues, potentially: (a) adjust the defense's hyperparameters to retain numerical stability; (b) replace the offending component with a differentiable or stable counterpart, e.g., substitute the low-rank approximation of SVD-GCN [12] with a suitable differentiable alternative; or (c) remove components, e.g., drop the "hard" filtering of edges done in the preprocessing of Soft-Median-GDC [17]. These considerations also include poisoning attacks, where one also needs to pay attention to all components of the training procedure. For example, we ignore the nuclear norm loss term in the training of ProGNN [30] to obtain the meta-gradient. Of course, keep the entire defense intact for its final evaluation on the found perturbations.

Step 5 – Adjust the attack loss. In previous works, the attack loss is often chosen to be the same as the training loss, i.e., the cross-entropy (CE). This is suboptimal since CE is not *consistent* according to the definition by Tramer et al. [44] – higher loss values do not indicate a stronger attack. Thus, we use a variant of the consistent Carlini-Wagner loss [4] for *local* attacks, namely the logit margin (LM), i.e., the logit difference between the ground truth class and most-likely non-true class. However, as discussed by Geisler et al. [17], for *global* attacks the mean LM across all target nodes is still suboptimal since it can "waste" budget on already misclassified nodes. Their tanh logit margin (TLM) loss resolves this issue. If not indicated otherwise, we either use TLM or the probability margin (PM) loss – a slight variant of LM that computes the margin after the softmax rather than before.

Step 6 – Tune the attack hyperparameters such as the number of PGD steps, the attack learning rate, the optimizer, etc. For example, for Metattack we observed that using the Adam optimizer [32] can weaken the attack and replacing it with SGD can increase the effectiveness.

Lessons learned. We provide a detailed description of each adaptive attack and the necessary actions to make it as strong as possible in § E. Here, we highlight some important recurring challenges that should be kept in mind when designing adaptive attacks. (1) Numerical issues, e.g., due to division by tiny numbers can lead to weak attacks, and we typically resolve them via clamping. (2) In some cases we observed that for PGD attacks it is beneficial to clip the gradients to stabilize the adversarial optimization. (3) For a strong attack it is essential to tune its hyperparameters. (4) Relaxing non-differentiable components and deactivating operations that filter edges/embeddings based on a threshold in order to obtain gradients for every edge is an effective strategy. (5) If the success of evasion-poisoning transfer depends on a fixed random initialization (see § J), it helps to use multiple clean auxiliary models trained with different random seeds for the PGD attack – in each PGD step we choose one model randomly. (6) Components that make the optimization more difficult but barely help the defense can be safely deactivated. (7) It is sometimes beneficial to control the randomness in the training loop of Meta-PGD. (8) For Meta-PGD it can help to initialize the attack with non-zero perturbations and e.g., use the perturbed graph of a different attack.

Example 1 – SVD-GCN. To illustrate the attack process (especially steps 3 and 4) we present a case study of how we construct an adaptive attack against SVD-GCN. Gradient-free attacks like Nettack do not work well here as they waste budget on adversarial edges which are filtered out by the low-rank approximation (LRA). Moreover, to the demise of gradient-based attacks, the gradients of the adjacency matrix are very unstable due to the SVD and thus less useful. Still, we start with a gradient-based attack as it is easier to adapt, specifically FGA, whose quick runtime enables rapid prototyping as it requires only a single gradient calculation. To replace the LRA with a function whose gradients are better behaved, we first decompose the perturbed adjacency matrix $\tilde{\mathbf{A}} = \mathbf{A} + \delta\mathbf{A}$ and, thus, only need gradients for $\delta\mathbf{A}$. Next, we notice that the eigenvectors of $\mathbf{A}$ usually have few large components. Perturbations along those principal dimensions are representable by the eigenvectors, hence most likely are neither filtered out nor impact the eigenvectors. Knowing this, we approximate the LRA in a tractable manner by element-wise multiplication of $\delta\mathbf{A}$ with weights that quantify how well an edge aligns with the principal dimensions (details in § E). In short we replace $\mathrm{LRA}(\mathbf{A} + \delta\mathbf{A})$ with $\mathrm{LRA}(\mathbf{A}) + \delta\mathbf{A} \circ \mathrm{Weight}(\mathbf{A})$, which admits useful gradients. This approach carries over to other attacks such as Nettack – we can incorporate the weights into its score function to avoid selecting edges that will be filtered out.

Example 2 – ProGNN. While we approached SVD-GCN with a theoretical insight, breaking a composite defense like ProGNN requires engineering and tinkering. When attacking ProGNN with PGD and transferring the perturbations to poisoning we observe that the perturbations are only effective if the model is trained with the same random seed. This over-sensitivity can be avoided by

employing lesson (5) in § 4. As ProGNN is very expensive to train due to its nuclear norm regularizer, we drop that term when training the set of auxiliary models without hurting attack strength. For unrolling the training we again drop the nuclear norm regularizer since it is non-differentiable. Sometimes PGD does not find a state with high attack loss, which can be alleviated by random restarts. As Meta-PGD optimization quickly stalls, we initialize it with a strong perturbation found by Meta-PGD on GCN. All of these tricks combined are necessary to successfully attack ProGNN.

Effort. Breaking Jaccard-GCN (and SVD-GCN) required around half an hour (resp. three days) of work for the initial proof of concept. Some other defenses require various adjustments that need to be developed over time, but reusing those can quickly break even challenging defenses. It is difficult to quantify this effort, but it can be greatly accelerated by adopting our lessons learned in § 4. In any case, we argue that authors proposing a new defense must put in reasonable effort to break it.

5 Evaluation of adaptive attacks

First, we provide details on the experimental setup and used metrics. We then report the main results and findings. We refer to § A for details on the base attacks, including our Greedy Brute Force and Meta-PGD approaches. We provide the code, configurations, and a collection of perturbed graphs on the project website linked on the first page.

Setup. We use the two most widely used datasets in the literature, namely Cora ML [2] and Citeseer [19] (details in § F). Unfortunately, larger datasets are barely possible since most defenses are not very scalable. Still, in § N, we discuss scalability and apply an adaptive attack to arXiv (170k nodes) [23]. We repeat the experiments for five different data splits (10% training, 10% validation, 80% testing) and report the means and variances. We use an internal cluster with Nvidia GTX 1080Ti GPUs. Most experiments can be reproduced within a few hours. However, the experiments with ProGNN and GRAND will likely require several GPU days.

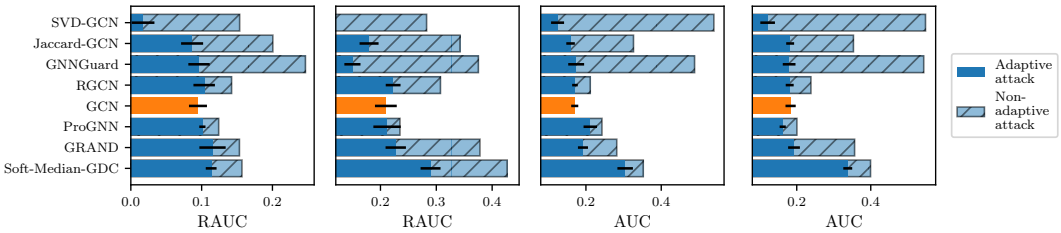
Defense hyperparameters. When first attacking the defenses, we observed that many exhibit poor robustness using the hyperparameters provided by their authors. To not accidentally dismiss a defense as non-robust, we tune the hyperparameters such that the clean accuracy remains constant but the robustness w.r.t. adaptive attacks is improved. Still, we run all experiments on the untuned defenses as well to confirm we achieve this goal. In the same way, we also tune the GCN model, which we use as a reference to asses whether a defense has merit. We report the configurations and verify the success of our tuning in § H.

Attacks and budget. In the *global* setting, we run the experiments for budgets Δ of up to 15% of the total number of edges in the dataset. Due to our (R)AUC metric (see below), we effectively focus on only the lower range of evaluated budgets. We apply FGA and PGD [53] for evasion. For poisoning, we transfer the found perturbations and also run Metattack [66] and our Meta-PGD. Recall that where necessary, we adapt the attacks to the defenses as outlined in § 4 and detailed in § E.

In the *local* setting, we first draw sets of 20 target nodes per split with degrees 1, 2, 3, 5, 8-10, and 15-25 respectively (total of 120 nodes). This enables us to study how the attacks affect different types of nodes – lower degree nodes are often conjectured to be less robust (see also § K). We then run the experiments for relative budgets Δ of up to 200% of the target node's degree. For example, if a node has 10 neighbors, and the budget $\Delta = 70\%$ then the attacker can change up to $10 \cdot 0.7 = 7$ edges. This commonly used setup ensures that we treat both low and high-degree nodes fairly. We use Nettack [67], FGA, PGD, and our greedy brute force attack for evasion. For poisoning, we only transfer the found perturbations. Again, we adapt the attacks to the defenses if necessary.

In alignment with our threat model, we evaluate each found perturbation by the test set accuracy it achieves (*global*) or the ratio of target nodes that remain correctly classified (*local*). For each budget, we choose the strongest attack among all attempts (e.g., PGD, Metattack, Meta-PGD). This gives rise to an envelope curve as seen in Fig. 3. We also include lower budgets as attempts, i.e., we enforce the envelope curve to be monotonically decreasing.

We introduce a rich set of attack characteristics by also transferring the perturbations supporting the envelope curve to every other defense. These transfer attacks then also contribute to the final envelope curve of each defense, but in most cases their contribution is marginal.



(a) Global, Poisoning (b) Global, Evasion (c) Local, Poisoning (d) Local, Evasion

Figure 2: Adaptive vs. non-adaptive attacks with budget-agnostic (R)AUC on Cora ML (c.f. Fig. 1). SVD-GCN (b) is disastrously broken – our adaptive attacks reach <0.02 (not visible). § F for Citeseer.

Non-adaptive attacks. We call any attack "non-adaptive" that is not aware of any changes made to the model (including defense mechanisms). Where we report results for a non-adaptive attack (e.g., Fig. 1 or Fig. 2), we specifically refer to an attack performed on a (potentially linearlized) GCN with commonly used hyperparameters (i.e., untuned). We then apply the perturbed adjacency matrix to the actual defense. In other words, we transfer the adversarial perturbation from a GCN. For our *local* non-adaptive attack, we always use Nettack. In contrast, for our *global* non-adaptive attack, we apply all attacks listed above, and then transfer for each budget the attack which is strongest against the GCN. Due to this ensemble of attacks, our global non-adaptive attack is expected to be slightly stronger than the non-adaptive attacks in most other works.

Area Under the Curve (AUC). An envelope curve gives us a detailed breakdown of the empirical robustness of a defense for different adversarial budgets. However, it is difficult to compare different attacks and defenses by only visually comparing their curves in a figure (e.g., see Fig. 4). Therefore, in addition to this breakdown per budget, we summarize robustness using the Area Under the Curve (AUC), which is independent of a specific choice of budget Δ and also punishes defenses that achieve robustness by trading in too much clean accuracy. Intuitively higher AUCs indicate more robust models, and conversely, lower AUCs indicate stronger attacks.

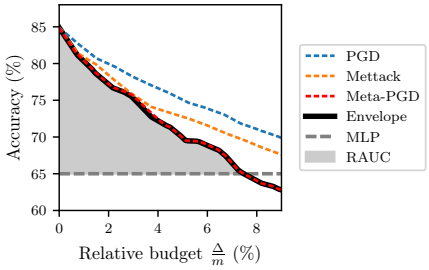
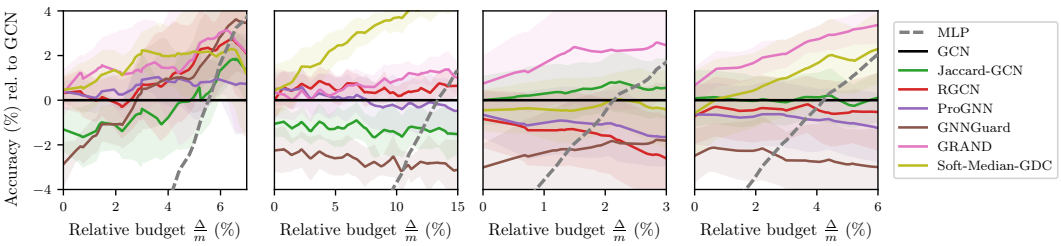


Figure 3: The dotted lines show the test set accuracy per budget after three global poisoning attacks against a tuned GCN on Cora ML. Taking the envelope gives the solid black robustness curve. The dashed gray line denotes the accuracy of an MLP. The shaded area is the RAUC.

As our *local* attacks break virtually all target nodes within our conservative maximum budget (see § F), taking the AUC over all budgets conveniently measures how quick this occurs. However, for *global* attacks, the test set accuracy continues to decrease for unreasonably large budget, and it is unclear when to stop. To avoid having to choose a maximum budget, we wish to stop when discarding the entire tainted graph becomes the better defense. This is fulfilled by the area between the envelope curve and the line signifying the accuracy of an MLP – a model that is oblivious to the graph structure, at the expense of a substantially lower clean accuracy than a GNN. We call this metric Relative AUC (RAUC) and illustrate it in Fig. 3. More formally, $\mathrm{RAUC}(c) = \int_0^{b_0} (c(b) - a_{\mathrm{MLP}}) \mathrm{d}b$ s.t. $b \lessgtr b_0 \implies c(b) \gtrless a_{\mathrm{MLP}}$ where $c(\cdot)$ is a piecewise linear robustness per budget curve, and a_{MLP} is the accuracy of the MLP baseline. We normalize the RAUC s.t. 0% is the performance of an MLP and 100% is the optimal score (i.e., 100% accuracy).

Finding 1 – Our adaptive attacks lower robustness by 40% on average. In Fig. 2 we compare non-adaptive attacks, the current standard to evaluate defenses, with our adaptive attacks which we propose as a new standard. The achieved (R)AUC in each case drops on average by 40% (similarly for Citeseer, see § F). In other words, the reported robustness in the original works proposing a defense is roughly 40% too optimistic. We confirm a statistically significant drop ($p < 0.05$) with a one-sided t-test in 85% of all cases. Considering adversarial accuracy for (small) fixed adversarial budget (Fig. 1) instead of the summary (R)AUC over all budgets tells the same story: non-adaptive attacks are too weak to be reliable indicators of robustness and adaptive attacks massively shrink the alleged robustness gains.



(a) Cora ML, Pois. (b) Cora ML, Evas. (c) Citeseer, Pois. (d) Citeseer, Evas.

Figure 4: Difference (defense – undefended GCN) of adversarial accuracy for the strongest global attack per budget. Almost half of the defenses perform worse than the GCN. We exclude SVD-GCN since it is catastrophically broken and plotting it would make the other defenses illegible (accuracy <24% already for a budget of 2% on Cora ML). Absolute numbers in § F.

Finding 2 – Structural robustness of GCN is not easily improved. In Fig. 4 (global) and Fig. 5 (local) we provide a more detailed view for different adversarial budgets and different graphs. For easier comparison we show the accuracy relative to the undefended GCN baseline. Overall, the decline is substantial. Almost half of the examined defenses perform worse than GCN and most remaining defenses neither meaningfully improve nor lower the robustness (see also Fig. 1 and Fig. 3). GRAND and Soft-Medoid-GCN retain robustness in some settings, but the gains are smaller than reported.

Finding 3 – Defense effectiveness depends on dataset. As we can see in Fig. 4 and Fig. 5, our ability to circumvent specific defenses tends to depend on the dataset. It appears that some defenses are more suited for different datasets. For example, GRAND seems to be a good choice for Citeseer while it is not as strong on Cora ML. The results for local attacks (Fig. 5) paint a similar picture, here we see that Cora ML is more difficult to defend. This points to another potentially problematic pitfall: most defenses are developed only using these two datasets as benchmarks. Is robustness even worse on other graphs? We leave this question for future work.

Finding 4 – No trade-off between accuracy and robustness for structure perturbations. Instead, Fig. 6 shows that defenses with high clean accuracy also exhibit high RAUC, i.e., are more robust against our attacks. This appears to be in contrast to the image domain [45]. However, we cannot exclude that future more powerful defenses might manifest this trade-off in the graph domain.

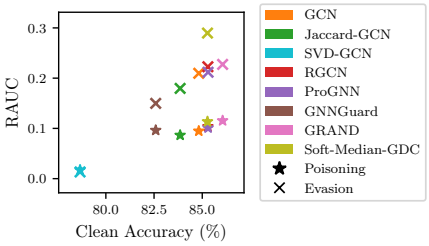
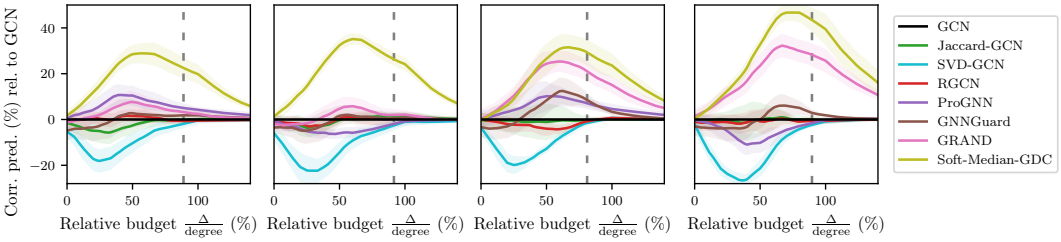


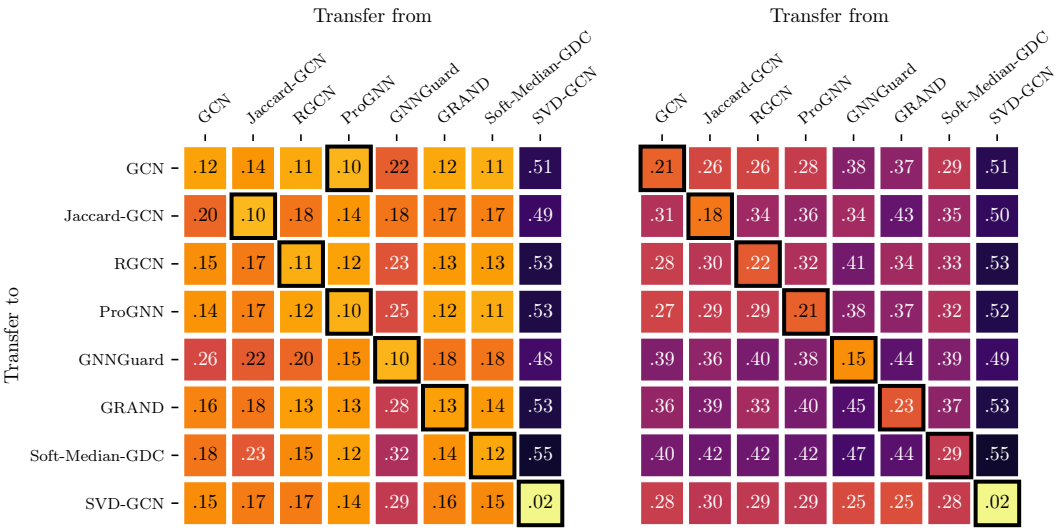
Figure 6: Model accuracy vs. RAUC of the strongest global attacks on Cora ML. We do not observe a robustness accuracy trade-off, but even find models with higher accuracy to be more robust.

Additional analysis. During this project, we generated a treasure trove of data. We perform a more in-depth analysis of our attacks in the appendix. First, we study how node degree affects attacks (see § K). For local attacks, the required budget to misclassify a node is usually proportional to the node's degree. Global attacks tend to be



(a) Cora ML, Pois. (b) Cora ML, Evas. (c) Citeseer, Pois. (d) Citeseer, Evas.

Figure 5: Difference (defense – undefended GCN) of fraction of correct predictions for the strongest local attack per budget. Most defenses show no or only marginal gain in robustness. The dashed vertical line shows where 95% of nodes for a GCN are misclassified on average. Abs. numbers in § F.



(a) Poisoning (b) Evasion

Figure 7: RAUC for the transfer of the strongest global adaptive attacks on Cora ML between models. The columns contain the models for which the adaptive attacks were created. The rows contain the RAUC after the transfer. With only one exception, adaptive attacks (diagonal) are most effective.

oblivious to degree and uniformly break nodes. Next, we perform a breakdown of each defense in terms of the sensitivity to different attacks (see § I). In short, global attacks are dominated by PGD for evasion and Metattack/Meta-PGD for poisoning with the PM or TLM loss. For local, our greedy brute-force is most effective, rarely beaten by PGD and Nettack. Finally, we analyze the properties of the adversarial edges in terms of various graph statistics such as edge centrality and frequency spectra (see § L § M).

6 Robustness unit test

Next we systematically study how well the attacks transfer between defenses, as introduced in the *attacks and budget* paragraph in § 5. In Fig. 7, we see that in 15 out of 16 cases the adaptive attack is the most effective strategy (see main diagonal). However for many defenses, there is often a source model or ensemble of source models (for the latter see § G) which forms a strong transfer attack.

Motivated by the effectiveness of transfer attacks (especially if transferring from ProGNN [30]), we suggest this set of perturbed graphs to be used as a bare minimum robustness unit test: one can probe a new defense by testing against these perturbed graphs, and if there exists at least one that diminishes the robustness gains, we can immediately conclude that the defense is not robust in the worst-case – without the potentially elaborate process of designing a new adaptive attack. We provide instructions on how to use this collection in the accompanying code.

Nevertheless, we cannot stress enough that this collection does not replace a properly developed adaptive attack. For example, if one would come up with SVD-GCN and would use our collection (excluding the perturbed graphs for SVD-GCN) the unit test would partially pass. However, as we can see in e.g., Fig. 2, SVD-GCN can be broken with an – admittedly very distinct – adaptive attack.

7 Related work

Excluding attacks on undefended GNNs, previous works studying adaptive attacks in the graph domain are scarce. The recently proposed graph robustness benchmark [62] also only studies transfer attacks. Such transfer attacks are so common in the graph domain that their usage is often not even explicitly stated, and we find that the perturbations are most commonly transferred from Nettack or Metattack (both use a linearized GCN). Other times, the authors of a defense only state that they use PGD [53] (aka "topology attack") without further explanations. In this case, the authors most

certainly refer to a PGD transfer attack on a GCN proxy. They almost never apply PGD to their actual defense, which would yield an adaptive attack (but possibly weak, see § 4 for guidance).

An exception where the defense authors study an adaptive attack is SVD-GCN [12]. Their attack collects the edges flipped by Nettack in a difference matrix $\delta\mathbf{A}$, replaces its most significant singular values and vectors with those from the clean adajcency matrix $\mathbf{A}$, and finally adds it to $\mathbf{A}$. Notably, this yields a dense continuous perturbed adjacency matrix. While their SVD-GCN is susceptible to these perturbations, the results however do not appear as catastrophic as with our adaptive attacks, despite their severe violation of our threat model (see § 2). Geisler et al. [17] are another exception where gradient-based greedy and PGD attacks are directly applied to their Soft-Median-GDC defense, making them adaptive. Still, our attacks manage to further reduce their robustness estimate.

8 Discussion

We hope that the adversarial learning community for GNNs will reflect on the bitter lesson that evaluating adversarial robustness is not trivial. We show that on average adversarial robustness estimates are overstated by 40%. To ease the transition into a more reliable regime of robustness evaluation for GNNs we share our recipe for successfully designing strong adaptive attacks.

Using adaptive (white-box) attacks is also interesting from a security perspective. If a model successfully defends such strong attacks, it is less likely to have remaining attack vectors for a real-world adversary. Practitioners can use our methodology to evaluate their models in hope to avoid an arms race with attackers. Moreover, the white-box assumption lowers the chance that real-world adversaries can leverage our findings, as it is unlikely that they have perfect knowledge.

We also urge for caution since the attacks only provide an upper bound (which with our attacks is now 40% tighter). Nevertheless, we argue that the burden of proof that a defense is truly effective should lie with the authors proposing it. Following our methodology, the effort to design a strong adaptive attack is reduced, so we advocate for adaptive attacks as the gold-standard for future defenses.

Acknowledgments and Disclosure of Funding

This research was supported by the Helmholtz Association under the joint research school "Munich School for Data Science – MUDS".

References

[1] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International Conference on Machine Learning, ICML*, 2018.

[2] Aleksandar Bojchevski and Stephan Günnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. In *International Conference on Learning Representations, ICLR*, 2018.

[3] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *ACM Workshop on Artificial Intelligence and Security, AISec*, 2017.

[4] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy*, 2017.

[5] Heng Chang, Yu Rong, Tingyang Xu, Yatao Bian, Shiji Zhou, Xin Wang, Junzhou Huang, and Wenwu Zhu. Not all low-pass filters are robust in graph convolutional networks. In *Advances in Neural Information Processing Systems, NeurIPS*, 2021.

[6] J. Chen, X. Lin, H. Xiong, Y. Wu, H. Zheng, and Q. Xuan. Smoothing adversarial training for GNN. *IEEE Transactions on Computational Social Systems*, 8(3), 2020.

[7] Liang Chen, Jintang Li, Qibiao Peng, Yang Liu, Zibin Zheng, and Carl Yang. Understanding structural vulnerability in graph convolutional networks. In *International Joint Conference on Artificial Intelligence, IJCAI*, 2021.

[8] Lingwei Chen, Xiaoting Li, and Dinghao Wu. Enhancing robustness of graph convolutional networks via dropping graph connections. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, ECML PKDD*, 2021.

[9] Zhijie Deng, Yinpeng Dong, and Jun Zhu. Batch virtual adversarial training for graph convolutional networks. In *Workshop on Learning and Reasoning with Graph-Structured Representations at the International Conference on Machine Learning, ICML*, 2019.

[10] Dongsheng Duan, Lingling Tong, Yangxi Li, Jie Lu, Lei Shi, and Cheng Zhang. AANE: Anomaly aware network embedding for anomalous link detection. In *IEEE International Conference on Data Mining, ICDM*, 2020.

[11] Pantelis Elinas, Edwin V. Bonilla, and Louis Tiao. Variational inference for graph convolutional networks in the absence of graph data and adversarial settings. In *Advances in Neural Information Processing Systems, NeurIPS*, 2020.

[12] Negin Entezari, Saba A. Al-Sayouri, Amirali Darvishzadeh, and Evangelos E. Papalexakis. All you need is low (rank): Defending against adversarial attacks on graphs. In *ACM International Conference on Web Search and Data Mining, WSDM*, 2020.

[13] Boyuan Feng, Yuke Wang, Z. Wang, and Yufei Ding. Uncertainty-aware attention graph neural network for defending adversarial attacks. In *AAAI Conference on Artificial Intelligence*, 2021.

[14] Fuli Feng, Xiangnan He, Jie Tang, and Tat-Seng Chua. Graph adversarial training: Dynamically regularizing based on graph structure. *IEEE Transactions on Knowledge and Data Engineering*, 33(6), 2021.

[15] Wenzheng Feng, Jie Zhang, Yuxiao Dong, Yu Han, Huanbo Luan, Qian Xu, Qiang Yang, Evgeny Kharlamov, and Jie Tang. Graph random neural network for semi-supervised learning on graphs. In *International Conference on Machine Learning, ICML*, 2021.

[16] Simon Geisler, Daniel Zügner, and Stephan Günnemann. Reliable graph neural networks via robust aggregation. In *Advances in Neural Information Processing Systems, NeurIPS*, 2020.

[17] Simon Geisler, Tobias Schmidt, Hakan Sirin, Daniel Zügner, Aleksandar Bojchevski, and Stephan Günnemann. Robustness of graph neural networks at scale. In *Advances in Neural Information Processing Systems, NeurIPS*, 2021.

[18] Simon Geisler, Johanna Sommer, Jan Schuchardt, Aleksandar Bojchevski, and Stephan Günnemann. Generalization of neural combinatorial solvers through the lens of adversarial robustness. In *International Conference on Learning Representations, ICLR*, 2022.

[19] C. Lee Giles, Kurt D. Bollacker, and Steve Lawrence. CiteSeer: An automatic citation indexing system. In *ACM Conference on Digital Libraries*, 1998.

[20] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning, ICML*, 2017.

[21] Stephan Günnemann. Graph neural networks: Adversarial robustness. In Lingfei Wu, Peng Cui, Jian Pei, and Liang Zhao, editors, *Graph Neural Networks: Foundations, Frontiers, and Applications*, chapter 8, . Springer, 2021.

[22] Weibo Hu, Chuan Chen, Yaomin Chang, Zibin Zheng, and Yunfei Du. Robust graph convolutional networks with directional graph adversarial training. *Applied Intelligence*, 51(11), 2021.

[23] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *Advances in Neural Information Processing Systems, NeurIPS*, 2020.

[24] Vassilis N. Ioannidis and Georgios B. Giannakis. Edge dithering for robust adaptive graph convolutional networks. In *AAAI Conference on Artificial Intelligence*, 2020.

[25] Vassilis N. Ioannidis, Antonio G. Marques, and Georgios B. Giannakis. Tensor graph convolutional networks for multi-relational and robust learning. *IEEE Transactions on Signal Processing*, 68, 2020.

[26] Vassilis N. Ioannidis, Dimitris Berberidis, and Georgios B. Giannakis. Unveiling anomalous nodes via random sampling and consensus on graphs. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, 2021.

[27] Hongwei Jin and Xinhua Zhang. Latent adversarial training of graph convolution networks. In *Workshop on Learning and Reasoning with Graph-Structured Representations at the International Conference on Machine Learning, ICML*, 2019.

[28] Hongwei Jin and Xinhua Zhang. Robust training of graph convolutional networks via latent perturbation. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, ECML PKDD*, 2021.

[29] Ming Jin, Heng Chang, Wenwu Zhu, and Somayeh Sojoudi. Power up! Robust graph convolutional network against evasion attacks based on graph powering. In *AAAI Conference on Artificial Intelligence*, 2021.

[30] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. Graph structure learning for robust graph neural networks. In *ACM International Conference on Knowledge Discovery and Data Mining, SIGKDD*, 2020.

[31] Wei Jin, Tyler Derr, Yiqi Wang, Yao Ma, Zitao Liu, and Jiliang Tang. Node similarity preserving graph convolutional networks. In *ACM International Conference on Web Search and Data Mining, WSDM*, 2021.

[32] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations, ICLR*, 2015.

[33] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations, ICLR*, 2017.

[34] Jintang Li, Tao Xie, Chen Liang, Fenfang Xie, Xiangnan He, and Zibin Zheng. Adversarial attack on large scale graph. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[35] Yaxin Li, Wei Jin, Han Xu, and Jiliang Tang. Deeprobust: A pytorch library for adversarial attacks and defenses. *arXiv preprint arXiv:2005.06149*, 2020.

[36] Xiaorui Liu, Jiayuan Ding, Wei Jin, Han Xu, Yao Ma, Zitao Liu, and Jiliang Tang. Graph neural networks with adaptive residual. In *Advances in Neural Information Processing Systems, NeurIPS*, 2021.

[37] Xiaorui Liu, Wei Jin, Yao Ma, Yaxin Li, Hua Liu, Yiqi Wang, Ming Yan, and Jiliang Tang. Elastic graph neural networks. In *International Conference on Machine Learning, ICML*, 2021.

[38] Dongsheng Luo, Wei Cheng, Wenchao Yu, Bo Zong, Jingchao Ni, Haifeng Chen, and Xiang Zhang. Learning to drop: Robust graph neural network via topological denoising. In *ACM International Conference on Web Search and Data Mining, WSDM*, 2021.

[39] Florence Regol, Soumyasundar Pal, Jianing Sun, Yingxue Zhang, Yanhui Geng, and Mark Coates. Node copying: A random graph model for effective graph sampling. *Signal Processing*, 192, 2022.

[40] Uday Shankar Shanthamallu, Jayaraman J. Thiagarajan, and Andreas Spanias. Uncertainty-matching graph neural networks to defend against poisoning attacks. In *AAAI Conference on Artificial Intelligence*, 2021.

[41] Ke Sun, Zhouchen Lin, Hantao Guo, and Zhanxing Zhu. Virtual adversarial training on graph convolutional networks in node classification. In *Chinese Conference on Pattern Recognition and Computer Vision, PRCV*, 2019.

[42] Xianfeng Tang, Yandong Li, Yiwei Sun, Huaxiu Yao, Prasenjit Mitra, and Suhang Wang. Transferring robustness for graph neural network against poisoning attacks. In *ACM International Conference on Web Search and Data Mining, WSDM*, 2020.

[43] Shuchang Tao, H. Shen, Q. Cao, L. Hou, and Xueqi Cheng. Adversarial immunization for certifiable robustness on graphs. In *ACM International Conference on Web Search and Data Mining, WSDM*, 2021.

[44] Florian Tramer, Nicholas Carlini, Wieland Brendel, and Aleksander Madry. On adaptive attacks to adversarial example defenses. In *Advances in Neural Information Processing Systems, NeurIPS*, 2020.

[45] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. In *International Conference on Learning Representations, ICLR*, 2019.

[46] Haibo Wang, Chuan Zhou, Xin Chen, Jia Wu, Shirui Pan, and Jilong Wang. Graph stochastic neural networks for semi-supervised learning. In *Advances in Neural Information Processing Systems, NeurIPS*, 2020.

[47] Yiwei Wang, Shenghua Liu, Minji Yoon, Hemank Lamba, Wei Wang, Christos Faloutsos, and Bryan Hooi. Provably robust node classification via low-pass message passing. In *IEEE International Conference on Data Mining, ICDM*, 2020.

[48] Huijun Wu, Chen Wang, Yuriy Tyshetskiy, Andrew Docherty, Kai Lu, and Liming Zhu. Adversarial examples for graph data: Deep insights into attack and defense. In *International Joint Conference on Artificial Intelligence, IJCAI*, 2019.

[49] Tailin Wu, Hongyu Ren, Pan Li, and Jure Leskovec. Graph information bottleneck. In *Advances in Neural Information Processing Systems, NeurIPS*, 2020.

[50] Yang Xiao, Jie Li, and Wengui Su. A lightweight metric defence strategy for graph neural networks against poisoning attacks. In *International Conference on Information and Communications Security, ICICS*, 2021.

[51] Hui Xu, Liyao Xiang, Jiahao Yu, Anqi Cao, and Xinbing Wang. Speedup robust graph structure learning with low-rank information. In *ACM International Conference on Information & Knowledge Management, CIKM*, 2021.

[52] Jiarong Xu, Yang Yang, Junru Chen, Chunping Wang, Xin Jiang, Jiangang Lu, and Yizhou Sun. Unsupervised adversarially-robust representation learning on graphs. In *AAAI Conference on Artificial Intelligence*, 2022.

[53] Kaidi Xu, Hongge Chen, Sijia Liu, Pin Yu Chen, Tsui Wei Weng, Mingyi Hong, and Xue Lin. Topology attack and defense for graph neural networks: An optimization perspective. In *International Joint Conference on Artificial Intelligence, IJCAI*, 2019.

[54] Kaidi Xu, Sijia Liu, Pin-Yu Chen, Mengshu Sun, Caiwen Ding, Bhavya Kailkhura, and Xue Lin. Towards an efficient and general framework of robust training for graph neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, 2020.

[55] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. In *Advances in Neural Information Processing Systems, NeruIPS*, 2020.

[56] Baoliang Zhang, Xiaoxin Guo, Zhenchuan Tu, and Jia Zhang. Graph alternate learning for robust graph neural networks in node classification. *Neural Computing and Applications*, 34 (11), 2022.

[57] Li Zhang and Haiping Lu. A feature-importance-aware and robust aggregator for gcn. In *ACM International Conference on Information & Knowledge Management, CIKM*, 2020.

[58] Xiang Zhang and Marinka Zitnik. GNNGuard: Defending graph neural networks against adversarial attacks. In *Advances in Neural Information Processing Systems, NeurIPS*, 2020.

[59] Yingxue Zhang, Sakif Hossain Khan, and Mark Coates. Comparing and detecting adversarial attacks for graph deep learning. In *Workshop on Representation Learning on Graphs and Manifolds at the International Conference on Learning Representations, ICLR*, 2019.

[60] Yingxue Zhang, Florence Regol, Soumyasundar Pal, Sakif Khan, Liheng Ma, and Mark Coates. Detection and defense of topological adversarial attacks on graphs. In *International Conference on Artificial Intelligence and Statistics, AISTATS*, 2021.

[61] Cheng Zheng, Bo Zong, Wei Cheng, Dongjin Song, Jingchao Ni, Wenchao Yu, Haifeng Chen, and Wei Wang. Robust graph representation learning via neural sparsification. In *International Conference on Machine Learning, ICML*, 2020.

[62] Qinkai Zheng, Xu Zou, Yuxiao Dong, Yukuo Cen, Da Yin, Jiarong Xu, Yang Yang, and Jie Tang. Graph robustness benchmark: Benchmarking the adversarial robustness of graph machine learning. In *Advances in Neural Information Processing Systems, NeurIPS*, 2021.

[63] Dingyuan Zhu, Peng Cui, Ziwei Zhang, and Wenwu Zhu. Robust graph convolutional networks against adversarial attacks. In *ACM International Conference on Knowledge Discovery and Data Mining, SIGKDD*, 2019.

[64] Jun Zhuang and Mohammad Al Hasan. Defending graph convolutional networks against dynamic graph perturbations via bayesian self-supervision. In *AAAI Conference on Artificial Intelligence*, 2022.

[65] Jun Zhuang and Mohammad Al Hasan. How does bayesian noisy self-supervision defend graph convolutional networks? *Neural Processing Letters*, 54(4), 2022.

[66] Daniel Zügner and Stephan Günnemann. Adversarial attacks on graph neural networks via meta learning. In *International Conference on Learning Representations, ICLR*, 2019.

[67] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *ACM International Conference on Knowledge Discovery and Data Mining, SIGKDD*, 2018.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See § 8.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See § 8.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See § 5.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See § 5, § H and provided code.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] All experiments are repeated for five random data splits.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See beginning of § 5.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes]
(b) Did you mention the license of the assets? [No]
(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] See beginning of § 5.
(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Attacks overview

In this section, we make the ensemble of attacks explicit and explain essential details. We then adapt these attack primitives to circumvent the defense mechanisms (see § E).

Global evasion attacks. The goal of a global attack is to provoke the misclassification of a large fraction of nodes (i.e., the test set) jointly, crafting a single perturbed adjacency matrix. For evasion, we use *(1) the Fast Gradient Attack (FGA)* and *(2) Projected Gradient Descent (PGD)*. In FGA, we calculate the gradient towards the entries of the clean adjacency matrix $\nabla_{\mathbf{A}} \ell_{\text{attack}}(f_{\theta^*}(\mathbf{A}, \mathbf{X}), \mathbf{y})$ and then flip the highest-ranked edges at once s.t. we exhaust the budget Δ. In contrast, PGD requires multiple gradient updates since it uses gradient ascent (see § 2 or explanation below for Meta-PGD). We deviate from the PGD implementation of Xu et al. [53] is two ways: (I) we adapt the initialization of the perturbation before the first attack gradient descent step and (II) we adjust the final sampling of $\tilde{\mathbf{A}}$. See below for more details.

Global poisoning attacks. We either (a) transfer the perturbation $\tilde{\mathbf{A}}$ found by evasion attack (1) or (2) and use it to poison training, or (b) differentiate through the training procedure by unrolling it, thereby obtaining a meta gradient. The latter approach is taken by both *(3) Metattack* [66] and *(4) our Meta-PGD*. Metattack greedily flips a single edge in each iteration and then obtains a new meta gradient at the changed adjacency matrix. In Meta-PGD, we follow the same relaxation as Xu et al. [53] (see below as well as § 2) and obtain meta gradients at the relaxed adjacency matrices. In contrast to the greedy approach of Metattack, Meta-PGD is able to revise early decisions later on.

Meta-PGD. Next, we explain the details of Meta-PGD and we present the pseudo code for reference in Algorithm A.1. Recall that the discrete edges are relaxed $\{0, 1\} \to [0, 1]$ and that the "weight" of the perturbation reflects the probability of flipping the respective edge.

Algorithm A.1 Meta-PGD

1: **Input:** Adjacency matrix $\mathbf{A}$, node features $\mathbf{X}$, labels $\mathbf{y}$, GNN $f_\theta(\cdot)$, loss ℓ_{attack}
2: **Parameters:** Budget Δ, iterations E, learning rates α_t
3: Initialize $\mathbf{P}_0 \in \mathbb{R}^{n \times n}$
4: **for** $t \in \{1, 2, \ldots, E\}$ **do**
5: Step $\mathbf{P}^{(t)} \leftarrow \mathbf{P}^{(t-1)} + \alpha_t \nabla_{\mathbf{P}^{(t-1)}} \left[\ell_{\text{attack}} \left(f(\mathbf{A} + \mathbf{P}^{(t-1)}, \mathbf{X};\, \theta = \text{train}(\mathbf{A} + \mathbf{P}^{(t-1)}, \mathbf{X}, \mathbf{y})), \mathbf{y} \right) \right]$
6: Projection $\mathbf{P}^{(t)} \leftarrow \Pi_{\|\mathbb{E}[\mathbf{A}+\mathbf{P}^{(t)}]-\mathbf{A}\|_0 \leq 2\Delta}(\mathbf{P}^{(t)})$
7: Sample $\tilde{\mathbf{A}}$ s.t. $\|\tilde{\mathbf{A}} - \mathbf{A}\|_0 \leq 2\Delta$
8: Return $\tilde{\mathbf{A}}$

In the first step of Meta-PGD, we initialize the perturbation (line 3). In contrast to Xu et al. [53]'s suggestion, we find that initializing the perturbation with the zero matrix can cause convergence issues. Hence, we alternatively initialize the perturbation with $\tilde{\mathbf{A}}$ from an attack on a different model (see also lesson learned #8 in § 4).

In each attack iteration, a gradient ascent step is performed on the relaxed perturbed adjacency matrix $\tilde{\mathbf{A}}^{(t-1)} = \mathbf{A} + \mathbf{P}^{(t-1)}$ (line 5). For obtaining the meta gradient through the training process, the training is unrolled. For example, with vanilla gradient descent for training $f_\theta(\mathbf{A}, \mathbf{X}) = f(\mathbf{A}, \mathbf{X}; \theta)$, the meta gradient resolves to

$$\nabla_{\mathbf{P}^{(t-1)}} \left(\ell_{\text{attack}} \left[f\Big(\mathbf{A} + \mathbf{P}^{(t-1)}, \mathbf{X}; \theta = \theta_0 - \eta \sum_{k=1}^{E_{\text{train}}} \nabla_{\theta_{k-1}} \ell_{\text{train}}[f(\mathbf{A} + \mathbf{P}^{(t-1)}, \mathbf{X}; \theta = \theta_{k-1}), \mathbf{y}]\Big), \mathbf{y} \right] \right) \tag{A.1}$$

with number of training epochs E_{train}, fixed training learning rate η, and parameters after (random) initialization θ_0. Notice that to obtain our variant of non-meta PGD, it suffices to replace the gradient computation in line 5 with $\nabla_{\mathbf{P}^{(t-1)}} \left[\ell_{\text{attack}}(f_{\theta^*}(\mathbf{A} + \mathbf{P}^{(t-1)}, \mathbf{X}), \mathbf{y}) \right]$.

Thereafter in line 6, the perturbation is projected such that in expectation the budget is obeyed, i.e., $\Pi_{\|\mathbb{E}[\mathbf{A}+\mathbf{P}^{(t)}]-\mathbf{A}\|_0 \leq 2\Delta}$. First, the projection clips $\mathbf{A} + \mathbf{P}^{(t-1)}$ to be in $[0, 1]$. If the budget is violated after clipping, it solves

$$\arg\min_{\hat{\mathbf{P}}^{(t)}} \|\hat{\mathbf{P}}^{(t)} - \mathbf{P}^{(t)}\|_2 \qquad \text{s.t.} \quad \mathbf{A} + \hat{\mathbf{P}}^{(t)} \in [0, 1]^{n \times n} \text{ and } \sum |\hat{\mathbf{P}}^{(t)}| \leq 2\Delta \tag{A.2}$$

After the last iteration (line 7), each element of $\mathbf{P}^{(t)}$ is interpreted as a probability and multiple perturbations are sampled accordingly. The strongest drawn perturbed adjacency matrix (in terms of

attack loss) is chosen as $\tilde{\mathbf{A}}$. Specifically, in contrast to [53], we sample $K = 100$ potential solutions that all obey the budget Δ and then choose the one that maximizes the attack loss ℓ_{attack}.

Local attacks. For local attacks we only run evasion attacks, and then transfer them to poisoning. This is common practice (e.g., see Zügner et al. [67] or Li et al. [34]). The attacks we use are *(1) FGA*, *(2) PGD*, *(3) Nettack [67]*, and a *(4) Greedy Brute Force* attack. Nettack greedily flips the best edges considering a linearized GCN, whose weights are either specially trained or taken from the attacked defense. In contrast, in each iteration, our Greedy Brute Force attack flips the current worst-case edge for the attacked model. It determines the worst-case perturbation by evaluating the model for every single edge flip. Notice that all examined models use two propagation steps, so we only consider all potential edges adjoining the target node or its neighbors[4]. Importantly, Greedy Brute Force is adaptive for any kind of model. Runtime-wise, the algorithm evaluates the attacked model $\mathcal{O}(\Delta n d)$ times with the number of nodes n and the degree of the target node d. We provide pseudo code in Algorithm A.2.

Algorithm A.2 Greedy Brute Force

Input: Target node i, adjacency matrix $\mathbf{A}$, node features $\mathbf{X}$, labels $\mathbf{y}$, GNN $f_\theta(\cdot)$, loss ℓ_{attack}
Parameter: Budget Δ
Initialize $\tilde{\mathbf{A}}^{(0)} = \mathbf{A}$
for $t \in \{1, 2, \ldots, \Delta\}$ **do**
  for potential edge e adjoining i or any of i's direct neighbors **do**
    Flip edge $\tilde{\mathbf{A}}^{(t)} \leftarrow \tilde{\mathbf{A}}^{(t-1)} \pm e$
    Remember best $\tilde{\mathbf{A}}^{(t)}$ in terms of $\ell_{\text{attack}}(f_{\theta^*}(\tilde{\mathbf{A}}^{(t)}, \mathbf{X}), \mathbf{y})$
    if node i is missclassifed **then**
      Return $\tilde{\mathbf{A}}^{(t)}$
  Recover best $\tilde{\mathbf{A}}^{(t)}$
Return $\tilde{\mathbf{A}}_\Delta$

Unnoticeability typically serves as a proxy to ensure that the label of an instance (here node) has not changed. In the image domain, it is widely accepted that a sufficiently small perturbation of the input image w.r.t. an L_p-norm is unnoticeable (and similarly for other threat models such as rotation). For graphs the whole subject of unnoticeability is more nuanced. The only constraint we use is the number of edge insertions/deletion, i.e., an L_0-ball around the clean adjacency matrix.

The only additional unnoticeability constraint proposed in the literature compares the clean and perturbed graph under a power law assumption on the node degrees [67]. However, we do not include such a constraint since (1) the degree distribution is only one (arbitrary) property to distinguish two graphs. (2) The degree distribution is a global property with an opaque relationship to the local class labels in node classification. (3) As demonstrated in Zügner & Günnemann [66], enforcing an indistinguishable degree distribution only has a negligible influence on attack efficacy, i.e., their gradient-based/adaptive attack conveniently circumvents this measure. Thus, we argue that enforcing such a constraint is similar to an additional (weak) defense measure and is not the focus of this work. Finally, since many defense (and attack) works in the literature considering node-classification (including the ones we study) also only use an L_0-ball constraint as a proxy for unnoticeability, we do the same for improved consistency. Out of scope are also other domains, like combinatorial optimization, where unnoticeability is not required since the true label of the perturbed instance is known [18].

[4] Due to GCN-like normalization (see § E), the three-hop neighbors need to be considered to be exhaustive. However, it is questionable if perturbing a neighbor three hops away is ever the strongest perturbation there is.

B Defense taxonomy

Next, we give further details behind our reasoning on how to categorize defenses for GNNs. Our taxonomy extends and largely follows Günnemann [21]'s. The three main categories are *improving the graph* (§ B.1), *improving the training* (§ B.2), and *improving the architecture* (§ B.3). We assign each defense to the category that fits best, even though some defenses additionally include ideas fitting into other categories as well. For the assignment of defenses see Table 1.

B.1 Improving the graph

With this category, we refer to all kinds of preprocessing of the graph. Alternatively, some approaches make the graph learnable with the goal of improved robustness. In summary, this category addresses changes that take place *prior* to the GNN (i.e., any message passing). We further distinguish *(1) unsupervised* and *(2) supervised* approaches.

Unsupervised. Any improvements that are not entangled with a learning objective, i.e., pure preprocessing, usually arising from clues found in the node features and graph structure. For example, Jaccard-GCN [48] filters out edges based on the Jaccard similarity of node features, while SVD-GCN [12] performs a low-rank approximation to filter out high-frequency perturbations. Most other approaches from this category exploit clues from features and structure simultaneously.

Supervised. These graph improvements are entangled with the learning objective by making the adjacency matrix learnable, often accompanied by additional regularization terms that introduce expert assumptions about robustness. For example, ProGNN [30] treats the adjacency matrix like a learnable parameter, and adds loss terms s.t. it remains close to the original adjacency matrix and exhibits properties which are assumed about clean graphs like low-rankness.

B.2 Improving the training

These approaches improve training – without changing the architecture – s.t. the learned parameters θ^* of the GNN exhibit improved robustness. In effect, the new training "nudges" a regular GNN towards being more robust. We distinguish *(1) robust training* and *(2) further training principles*.

Robust training. Alternative training schemes and losses which reward the correct classification of synthetic adversarial perturbations of the training data. With this category, Günnemann [21] targets both straightforward adversarial training and losses stemming from certificates (i.e., improving certifiable robustness). Neither approach is interesting to us: the former is discussed in § C, and the latter targets provable robustness which does not lend itself to empirical evaluation.

Further training principles. This category is distinct from robust training due to the lack of a clear mathematical definition of the training objective. It mostly captures augmentations [15, 29, 39, 42, 61] or alternative training schemes [5, 11, 55, 64] that encourage robustness. A simple example for such an approach is to pre-train the GNN weights on perturbed graphs [42]. Another recurring theme is to use multiple models during training and then, e.g., enforce consistency among them [5].

B.3 Improving the architecture

Even though there are some exceptions (see sub-category *(2) miscellaneous*), the recurring theme in this category is to somehow weight down the influence of some edges adaptively for each layer or message passing aggregation. We refer to this type of improved architecture with *(1) adaptively weighting edges*. We further distinguish between approaches that are *(a) rule-based*, *(b) probabilistic*, or use *(c) robust aggregation*.

Rule-based approaches typically use some metric [31, 58], alternative message passing [36, 37], or an auxiliary MLP [57] to filter out alleged adversarial edges. *Probabilistic* approaches either work with distributions in the latent space [63], are built upon probabilistic principles like Bayesian uncertainty quantification [13], or integrate sampling into the architecture and hence apply it also at inference time [8, 24, 25, 38]. *Robust aggregation* defenses replace the message passing aggregation (typically mean) with a more robust equivalent such as a trimmed mean, median, or soft median [7, 17]. In relation to the trimmed mean, in this category we include also other related approaches that come with some guarantees based on their aggregation scheme Wang et al. [47].

C On adversarial training defenses

The most basic form of adversarial training for structure perturbations aims to solve:

$$\min_{\theta} \max_{\mathbf{A}' \in \Phi(\mathbf{A})} \ell(f_\theta(\mathbf{A}', \mathbf{X}), \mathbf{y}) \tag{C.1}$$

Similarly to [44, 1, 4], we exclude defenses that build on adversarial training in our study for three reasons.

First, we observe that adversarial training requires knowing the clean $\mathbf{A}$. However, for poisoning, we would need to substitute $\mathbf{A}$ with an adversarially perturbed adjacency matrix $\tilde{\mathbf{A}}$. In this case, adversarial training aims to enforce adversarial generalization $\mathbf{A}' \in \Phi(\tilde{\mathbf{A}})$ for the adversarially perturbed adjacency matrix $\tilde{\mathbf{A}}$ – potentially even reinforcing the poisoning attack.

Second, an adaptive poisoning attack on adversarial training is very expensive as we need to unfold many adversarial attacks for a single training. Thus, designing truly adaptive poisoning attacks requires a considerable amount of resources. *Scaling* these attacks to such complicated training schemes is not the main objective of this work.

Third, adversarial training for structure perturbations on GNNs seems to be an unsolved question. So far, the robustness gains come from additional and orthogonal tricks such as self-training [53]. Hence, adversarial training for structure perturbations requires an entire paper on its own.

D On defenses against feature perturbations

As introduced in § 2, attacks may perturb the adjacency matrix $\mathbf{A}$, the feature matrix $\mathbf{X}$, or both. However, during our survey we found that few defenses tackle feature perturbations. Similarly, 6 out of the 7 defenses chosen by us mainly based on general popularity turn out to not consciously defend against feature perturbations.

The only exception is SVD-GCN [12], which also applies its low-rank approximation to the binary feature matrix. However, the authors do not report robustness under feature-only attacks; instead, they only consider mixed structure and feature attacks found by Nettack. Given the strong bias of Nettack towards structure perturbations, we argue that their experimental results do not confirm feature robustness. Correspondingly, in preliminary experiments we were not able to achieve considerable robustness gains of SVD-GCN compared to an undefended GCN – even with non-adaptive feature perturbations. If a non-adaptive attack is strong enough, there is not much merit in applying an adaptive attack.

To reiterate, due to the apparent scarcity of defenses apt against feature attacks, we decided to focus our efforts on structure attacks and defenses. However, new defenses considering feature perturbations should study robustness in the face of adaptive attacks – similarly to our work. In the following, we give some important hints for adaptive attacks using feature perturbations. We leave attacks that jointly consider feature and structure perturbations for future work due to the manifold open challenges, e.g., balancing structure and feature perturbations in the budget quantity.

Baseline. To gauge the robustness of defenses w.r.t. global attacks, we introduce the RAUC metric, which employs the accuracy of an MLP – which is perfectly robust w.r.t. structure perturbations – to determine the maximally sensible budget to include in the summary. As MLPs are however vulnerable to feature attacks, a different baseline model is required for this new setting. We propose to resolve this issue by using a label propagation approach, which is oblivious to the node features and hence perfectly robust w.r.t. feature perturbations.

Perturbations. The formulation of the set of admissible perturbations depends on what modality the data represents, which may differ between node features and graph edges. Convenient choices for continuous features are l-p-norms; in other cases, more complicated formulations are more appropriate. Accordingly, one has to choose an appropriate constrained optimization scheme.

E Examined adversarial defenses

In this section, we portray each defense and how we adapted the base attacks to each one. We refer to Table H.1 for the used hyperparameter values for each defense. We give the used attack parameters for a GCN below and refer to the provided code for the other defenses.

GCN. We employ an undefended GCN [33] as our baseline. A GCN first adds self loops to the adjacency matrix $\mathbf{A}$ and subsequently applies GCN-normalization, thereby obtaining $\mathbf{A}' = (\mathbf{D} + \mathbf{I})^{-\frac{1}{2}}(\mathbf{A} + \mathbf{I})(\mathbf{D} + \mathbf{I})^{-\frac{1}{2}}$ with the diagonal degree matrix $\mathbf{D} \in \mathbb{N}^{n \times n}$. Then, in each GCN layer it updates the hidden states $\mathbf{H}^{(l)} = \mathrm{dropout}(\sigma(\mathbf{A}'\mathbf{H}^{(l-1)}\mathbf{W}^{(l-1)} + \mathbf{b}^{(l-1)}))$ where $\mathbf{H}^{(0)} = \mathbf{X}$. We use the non-linear ReLU activation for intermediate layers. Dropout is deactivated in the last layer and we refer to the output before softmax activation as logits. We use Adam [32] to learn the model's parameters.

Attack. We do not require special tricks since the GCN is fully differentiable and does not come with defensive measures to consider. In fact, the off-the-shelf attacks we employ are tailored to a GCN. For PGD, we use $E = 200$ iterations, $K = 100$ samples, and a base learning rate of 0.1. For Meta-PGD, we only lower the base learning rate to 0.01 and add gradient clipping to 1 (w.r.t. global L_2-norm). For Metattack with SGD instead of Adam for training the GCN, we use an SGD learning rate of 1 and restrict the training to $E_{\text{train}} = 100$ epochs.

E.1 Jaccard-GCN

Defense. Additionally to a GCN, Jaccard-GCN [48] preprocesses the adjacency matrix. It computes the Jaccard coefficient of the binarized features for the pair of nodes of every edge, i.e., $\mathbf{J}_{ij} = \frac{\mathbf{X}_i \mathbf{X}_j}{\min\{\mathbf{X}_i + \mathbf{X}_j, 1\}}$. Then edges are dropped where $\mathbf{J}_{ij} \leq \epsilon$.

Adaptive attack. We do not need to adapt gradient-based attacks as the gradient is equal to zero for dropped edges. Straightforwardly, we adapt Nettack to only consider non-dropped edges. Analogously, we ignore these edges in the Greedy Brute Force attack for increased efficiency.

E.2 SVD-GCN

Defense. SVD-GCN [12] preprocesses the adjacency matrix with a low-rank approximation (LRA) for a fixed rank r, utilizing the Singular Value Decomposition (SVD) $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top \approx \mathbf{U}_r\mathbf{\Sigma}_r\mathbf{V}_r^\top = \mathbf{A}_r$. Note that the LRA is performed on $\mathbf{A}$ before adding self-loops and GCN-normalization (see above). Thereafter, the dense $\mathbf{A}_r$ is passed to the GCN as usual. Since $\mathbf{A}$ is symmetric and positive semi-definite, we interchangeably refer to the singular values/vectors also as eigenvalues/eigenvectors.

Adaptive attack. Unfortunately, the process of determining the singular vectors $\mathbf{U}_r$ and $\mathbf{V}_r$ is highly susceptible to small perturbations, and so is its gradient. Thus, we circumvent the need of differentiating the LRA.

We now explain the approach from a geometrical perspective. Each row of $\mathbf{A}$ (or interchangeably column as $\mathbf{A}$ is symmetric) is interpreted as coordinates of a high-dimensional point. The r most significant eigenvectors of $\mathbf{A}$ span an r-dimensional subspace, onto which the points are projected by the LRA. Adding or removing an adversarial edge (i, j) corresponds to moving the point $\mathbf{A}_i$ along dimension j, i.e., $\mathbf{A}_i \pm \mathbf{e}_j$ (vice-versa for $\mathbf{A}_j$). As hinted at in § 4, the r most significant eigenvectors of $\mathbf{A}$ turn out to usually have few large components. Thus, the relevant subspace is mostly aligned with only few dimensions.

Changes along the highest-valued eigenvectors are consequently preserved by LRA. To quantify how much exactly such a movement along a dimension j, i.e., $\mathbf{e}_j$, is preserved, we project the movement itself onto the subspace and extract the projected vector's j-th component. More formally, we denote the projection matrix onto the subspace as $\mathbf{P} = \sum_{k=0}^{r} \mathbf{v}_k \mathbf{v}_k^T$ where $\mathbf{v}_k$ are the eigenvectors of $\mathbf{A}$. We now score each dimension j with $(\mathbf{P}\mathbf{e}_j)_j = \mathbf{P}_{jj}$. Since the adjacency matrix is symmetric and rows and columns are hence exchangeable, we then symmetrize the scores $\mathbf{W}_{ij} = {}^{(\mathbf{P}_{ii} + \mathbf{P}_{jj})}/_{2}$.

Finally, we decompose the perturbed adjacency matrix $\tilde{\mathbf{A}} = \mathbf{A} + \delta\mathbf{A}$ and, thus, only need gradients for $\delta\mathbf{A}$. Using the approach sketched above, we now replace $\mathrm{LRA}(\mathbf{A} + \delta\mathbf{A}) \approx \mathrm{LRA}(\mathbf{A}) + \delta\mathbf{A} \circ \mathbf{W}$.

The weights $\mathbf{W}$ can also be incorporated into the Greedy Brute Force attack by dropping edges with weight < 0.2 and, for efficient early stopping, sort edges to try in order of descending weight. Similarly, Nettack's score function $s_{\text{struct}}(i, j)$ – which attains positive and negative values, while $\mathbf{W}$ is positive – can be wrapped to $s'_{\text{struct}}(i, j) = \log(\exp(s_{\text{struct}}(i, j)) \circ \mathbf{W}) = s_{\text{struct}}(i, j) + \log \mathbf{W}$.

Note that we assume that the direction of the eigenvectors remains roughly equal after perturbing the adjacency matrix. In practice, we find this assumption to be true. Intuitively, a change along the dominant eigenvectors should even reinforce their significance.

E.3 RGCN

Defense. The implementations of R(obust)GCN provided by the authors[5] and in the widespread DeepRobust [35] library[6] are both consistent, but diverge slightly from the paper [63]. We use and now present RGCN according to those reference implementations. Principally, RGCN models the hidden states as Gaussian vectors with diagonal variance instead of sharp vectors. In addition to GCN's $\mathbf{A}'$, a second $\mathbf{A}'' = (\mathbf{D} + \mathbf{I})^{-1}(\mathbf{A} + \mathbf{I})(\mathbf{D} + \mathbf{I})^{-1}$ is prepared to propagate the variances. The mean and variance of this hidden Gaussian distribution are initialized as $\mathbf{M}^{(0)} = \mathbf{V}^{(0)} = \mathbf{X}$. Each layer first computes an intermediate distributions given by $\hat{\mathbf{M}}^{(l)} = \mathrm{elu}(\mathrm{dropout}(\mathbf{M}^{(l-1)})\mathbf{W}_M^{(l-1)})$ and $\hat{\mathbf{V}}^{(l)} = \mathrm{relu}(\mathrm{dropout}(\mathbf{V}^{(l-1)})\mathbf{W}_V^{(l-1)})$. Then, attention coefficients $\boldsymbol{\alpha}^{(l)} = e^{-\gamma \hat{\mathbf{V}}^{(l)}}$ are calculated with the aim to subdue high-variance dimensions (where exponentiation is element-wise and γ is a hyperparameter). The final distributions are obtained with $\mathbf{M}^{(l)} = \mathbf{A}'\hat{\mathbf{M}}'^{(l)} \circ \boldsymbol{\alpha}^{(l)}$. Note the absence of bias terms. After the last layer, point estimates are sampled from the distributions via the reparameterization trick, i.e., scalars are sampled from a standard Gaussian and arranged in a matrix $\mathbf{R}$. These samples are then used to obtain the logits via $\mathbf{M}^{(L)} + \mathbf{R} \circ (\mathbf{V}^{(L)} + \epsilon)^{\frac{1}{2}}$ (where the square root applies element-wise and ϵ is a hyperparameter). Adam is the default optimizer. The loss is extended with the regularizer $\beta \sum_i \mathrm{KL}(\mathcal{N}(\hat{\mathbf{M}}_i^{(1)}, \mathrm{diag}(\hat{\mathbf{V}}_i^{(1)}))\|\mathcal{N}(\mathbf{0}, \mathbf{I}))$ (where β is a hyperparameter).

Adaptive attack. A direct gradient attack suffices for a strong adaptive attack. Only when unrolling the training procedure for Metattack and Meta-PGD, we increase hyperparameter ϵ from 10^{-8} to 10^{-2} to retain numerical stability.

E.4 ProGNN

Defense. We use and present Pro(perty)GNN [30] exactly following the implementation provided by the authors in their DeepRobust [35] library[6]. ProGNN learns an alternative adjacency matrix $\mathbf{S}$ that is initialized with $\mathbf{A}$. A regular GCN – which, as usual, adds self-loops and applies GCN-normalization – is trained using $\mathbf{S}$, which is simultaneously updated in every τ-th epoch. For that, first a gradient descent step is performed on $\mathbf{S}$ with learning rate η and momentum μ towards minimizing the principal training loss alongside two regularizers that measure deviation $\beta_1 \|\mathbf{S} - \mathbf{A}\|_F^2$ and feature smoothness $\frac{\beta_2}{2} \sum_{i,j} \mathbf{S}_{ij} \|\frac{\mathbf{X}_i}{\sqrt{d_i}} - \frac{\mathbf{X}_j}{\sqrt{d_j}}\|^2$ (where $d_i = \sum_j \mathbf{S}_{ij} + 10^{-3}$). Next, the singular value decomposition $\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$ of the updated $\mathbf{S}$ is computed, and $\mathbf{S}$ is again updated to be $\mathbf{U} \max(0, \boldsymbol{\Sigma} - \eta\beta_3)\mathbf{V}^T$ to promote low-rankness. Thereafter, $\mathbf{S}$ is again updated to be $\mathrm{sgn}(\mathbf{S}) \circ \max(0, |\mathbf{S}| - \eta\beta_4)$ to promote sparsity. Finally, the epoch's resulting $\mathbf{S}$ is obtained by clamping its elements between 0 and 1.

Adaptive attack. Designing an adaptive attack for ProGNN proved to be a challenging endeavor. We describe the collection of tricks in § 4's Example 2.

E.5 GNNGuard

Defense. We closely follow the authors' implementation[7] as it deviates from the formal definitions in the paper [58]. GNNGuard adopts a regular GCN and, before each layer, it adaptively weights down alleged adversarial edges. Thus, each layer has a unique propagation matrix $\mathbf{A}^{(l)}$ that is used instead of $\mathbf{A}'$.

[5] `https://github.com/ZW-ZHANG/RobustGCN` [6] `https://github.com/DSE-MSU/DeepRobust`
[7] `https://github.com/mims-harvard/GNNGuard`

GNNGuard's rule-based edge reweighting can be clustered into four consecutive steps: (1) the edges are reweighted based on the pair-wise cosine similarity $\mathbf{C}_{ij}^{(l)} = \frac{\mathbf{H}_i^{(l-1)} \cdot \mathbf{H}_j^{(l-1)}}{\|\mathbf{H}_i^{(l-1)}\| \|\mathbf{H}_j^{(l-1)}\|}$ according to $\mathbf{S}^{(l)} = \mathbf{A} \circ \mathbf{C}^{(l)} \circ \mathbb{I}[\mathbf{C}^{(l)} \geq 0.1]$, where edges with too dissimilar node embeddings are removed (see Iverson bracket $\mathbb{I}[\mathbf{C}^{(l)} \geq 0.1]$). Then, (2) the matrix is rescaled $\mathbf{\Gamma}_{ij}^{(l)} = \mathbf{s}_{ij}^{(l)} / \mathbf{s}_i^{(l)}$ with $\mathbf{s}_i^{(l)} = \sum_j \mathbf{S}_{ij}^{(l)}$ For stability, if $\mathbf{s}_i^{(l)} < \epsilon$, $\mathbf{s}_i^{(l)}$ is set to 1 (here ϵ is a small constant). Next, (3) self-loops are added and $\mathbf{\Gamma}^{(l)}$ is non-linarily transformed according to $\hat{\mathbf{\Gamma}}^{(l)} = \exp_{\neq 0}(\mathbf{\Gamma}^{(l)} + \operatorname{diag} {}^1\!/_{1 + \mathbf{d}^{(l)}})$, where $\exp_{\neq 0}$ only operates on nonzero elements and $\mathbf{d}_i^{(l)} = \|\mathbf{\Gamma}_i^{(l)}\|_0$ is the row-wise number of nonzero entries. Last, (4) the result is smoothed over the layers with $\mathbf{\Omega}^{(l)} = \sigma(\rho)\mathbf{\Omega}^{(l-1)} + (1 - \sigma(\rho))\hat{\mathbf{\Gamma}}^{(l)}$ with learnable parameter ρ and sigmoid function $\sigma(\cdot)$.

The resulting reweighted adjacency matrix $\mathbf{\Omega}^{(l)}$ is then GCN-normalized (without adding self-loops) and passed on to a GCN layer. Note that steps (1) to (3) are excluded from back-propagation during training. When comparing with the GNNGuard paper, one notices that among other deviations, we have omitted learnable edge pruning because it is disabled in the reference implementation.

Adaptive attack. The hyperparameter ϵ must be increased from 10^{-6} to 10^{-2} during the attack to retain numerical stability. In contrast to the reference implementation but as stated above, it is important to place the hard filtering step $\mathbb{I}[\mathbf{C}^{(l)} \geq 0.1]$ for $\mathbf{S}^{(l)}$ s.t. the gradient calculation w.r.t. $\mathbf{A}$ is not suppressed for these entries.

E.6 GRAND

Defense. The Graph Random Neural Network (GRAND) [15] model is the only defense from our selection that is not based on a GCN. First, $\mathbf{A}$ is endowed with self-loops and GCN-normalized to obtain $\mathbf{A}'$. Also, each row of $\mathbf{X}$ is l_1-normalized, yielding $\mathbf{X}'$. Next, rows from $\mathbf{X}'$ are randomly dropped with probability δ during training to generate a random augmentation, and $\mathbf{X}'$ is scaled by $1 - \delta$ during inference to compensate, thereby obtaining $\hat{\mathbf{X}}$. Those preprocessed node features are then propagated multiple times along the graph to get $\overline{\mathbf{X}} = \frac{1}{K+1} \sum_{k=0}^{K} \mathbf{A}'^k \hat{\mathbf{X}}$. Finally, dropout is applied once to $\overline{\mathbf{X}}$, and the result is plugged into a 2-layer MLP with dropout and ReLU activation to obtain class probabilities $\mathbf{Z}$. The authors also propose an alternative architecture using a GCN instead of an MLP, however, we do not explore this option since the MLP version is superior according to their own results.

GRAND is trained with Adam. The training loss comprises the mean of the cross-entropy losses of S model evaluations, thereby incorporating multiple random augmentations. Additionally, a consistency regularizer is added to enforce similar class probabilities across all evaluations. More formally, first the probabilities are averaged across all evaluations: $\overline{\mathbf{Z}} = \frac{1}{S} \sum_{s=1}^{S} \mathbf{Z}^{(s)}$. Next, each node's categorical distribution is sharpened according to a temperature hyperparameter T, i.e., $\overline{\mathbf{Z}}'_{ij} = \overline{\mathbf{Z}}_{ij}^{\frac{1}{T}} / \sum_c \overline{\mathbf{Z}}_{ic}^{\frac{1}{T}}$. The final regularizer penalizes the distance between the class probabilities and the sharpened averaged distributions, namely $\frac{\beta}{S} \sum_{s=1}^{S} \|\mathbf{Z}^{(s)} - \overline{\mathbf{Z}}'\|_F^2$.

Adaptive attack. When unrolling the training procedure for Metattack and Meta-PGD, to reduce the memory footprint, we reduce the number of random augmentations per epoch to 1, and we use a manual gradient calculation for the propagation operation. We also initialize Meta-PGD with a strong perturbation found by Meta-PGD on ProGNN. Otherwise, the attack has issues finding a perturbation with high loss; it presumably stalls in a local optimum. It is surprising that "only" initializing from GCN instead of ProGNN does not give a satisfyingly strong attack. Finally, we use the same random seed for every iteration of Metattack and Meta-PGD, as otherwise the constantly changing random graph augmentations make the optimization very noisy.

E.7 Soft-Median-GDC

Defense. The Soft-Median-GDC [17] deviates in two ways from a GCN: (1) it uses Personalized Page Rank (PPR) with restart probability $\alpha = 0.15$ to further preprocess the adjacency matrix after adding self-loops and applying GCN-normalization. The result is then sparsified using a row-wise top-k operation ($k = 64$). (2) the message passing aggregation is replaced with a robust estimator

called Soft-Median. From the perspective of node i, a GCN uses the message passing aggregation $\mathbf{H}_i^{(l)} = \mathbf{A}_i \mathbf{H}^{(l-1)}$ which can be interpreted as a weighted mean/sum. In Soft-Median-GDC, the "weights" $\mathbf{A}_i$ are replaced with a scaled version of $\mathbf{A}_i \circ \operatorname{softmax}\left(-\mathbf{c}/T\sqrt{d}\right)$. Here the vector $\mathbf{c}$ denotes the distance between hidden embedding of a neighboring node to the neighborhood-specific weighted dimension-wise median: $\mathbf{c}_i = \| \operatorname{Median}(\mathbf{A}_i, \mathbf{H}^{(l-1)}) - \mathbf{H}_i^{(l-1)} \|$. To keep the scale, these weights are scaled s.t. they sum up to $\sum \mathbf{A}_i$.

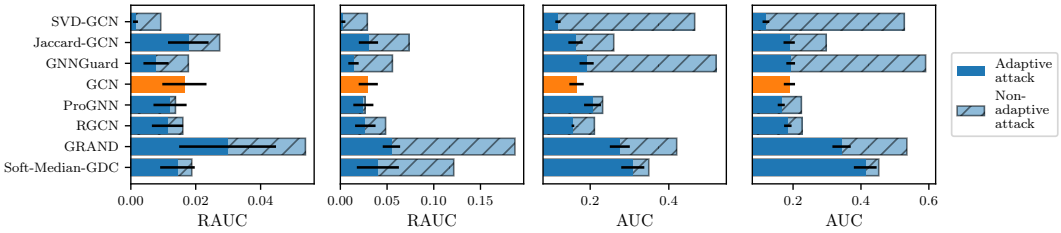
Adaptive attack. During gradient-based attacks, we adjust the $\mathbf{c}$ of every node s.t. it now captures the distance to all other nodes, not only neighbors. This of course modifies the values of $\mathbf{c}$, but is necessary to obtain a nonzero gradient w.r.t. to all candidate edges. We initialize PGD with a strong perturbation found by a similar attack on GCN, and initialize Meta-PGD with a perturbation from a similar attack on ProGNN (as with GRAND, using an attack against GCN as a base would be insufficient here).

F Evaluation of adaptive attacks

In Table F.1, we summarize the variants of the datasets we use, both of which we have precisely extracted from Nettack's code[8]. In Fig. F.1, we complement Fig. 2 and compare the (R)AUC of all defenses on Citeseer. The robustness estimates for the defenses on Citeseer are also much lower as originally reported. For completeness, we give absolute envelope curve plots for all settings and datasets as well as for higher budgets in Fig. F.2 and Fig. F.3 (compare with Fig. 4 and Fig. 5).

Table F.1: Statistics of the datasets we used. We measure homophily as the fraction of edges which connect nodes of the same class.

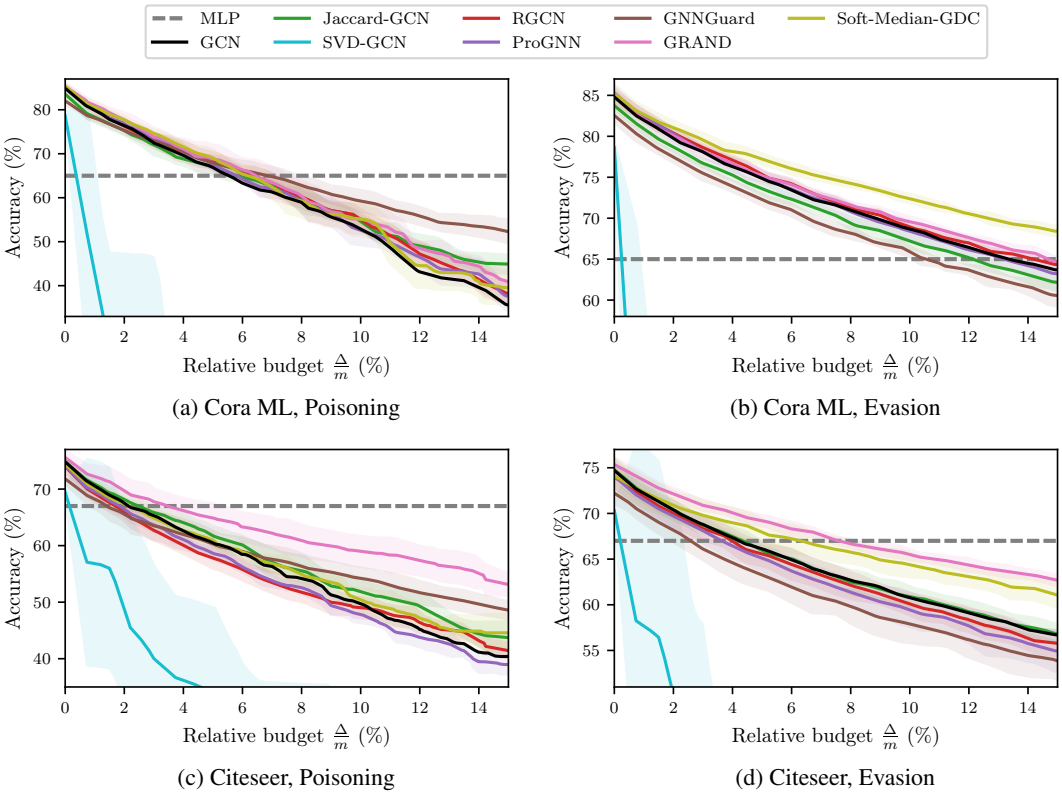
Dataset	Nodes	Undirected Edges	Features	Classes	Avg. Degree	Homophily
Cora ML [2]	2485	5069	1433	7	4.08	0.804
Citeseer [19]	2110	3668	3703	6	3.477	0.736



(a) Global, Poisoning (b) Global, Evasion (c) Local, Poisoning (d) Local, Evasion

Figure F.1: Variant of Fig. 2 for Citeseer.

[8] https://github.com/danielzuegner/nettack



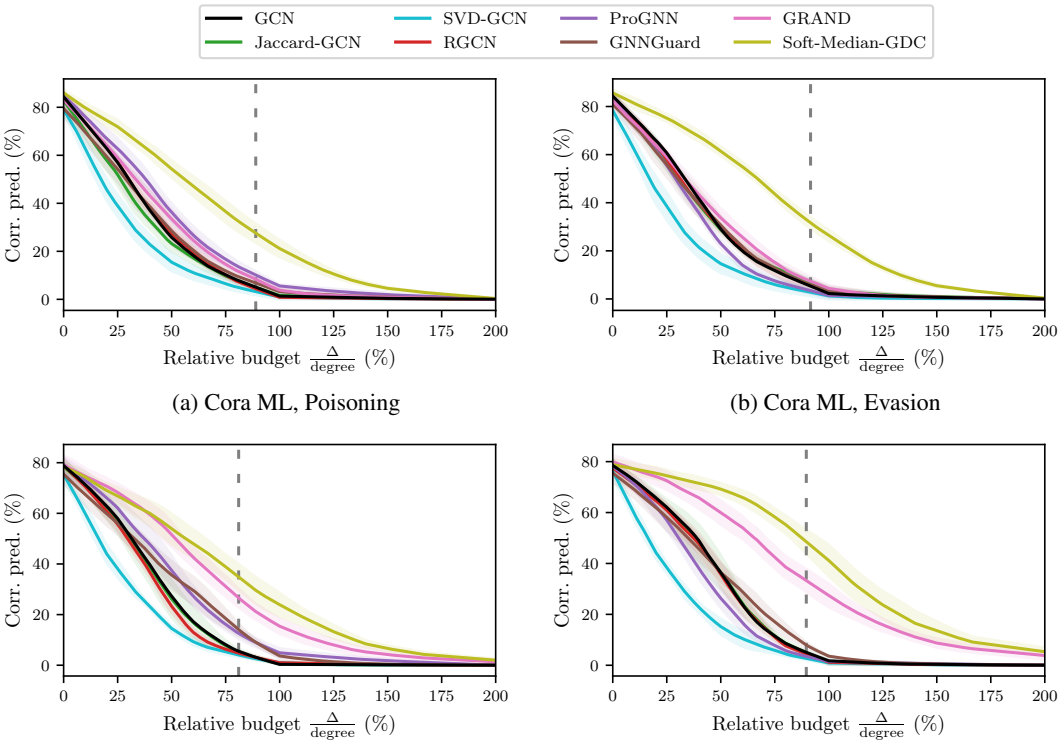
(a) Cora ML, Poisoning

(b) Cora ML, Evasion

(c) Citeseer, Poisoning

(d) Citeseer, Evasion

Figure F.2: Absolute variant of Fig. 4, showing relative budgets up to 15%.



(a) Cora ML, Poisoning

(b) Cora ML, Evasion

(c) Citeseer, Poisoning

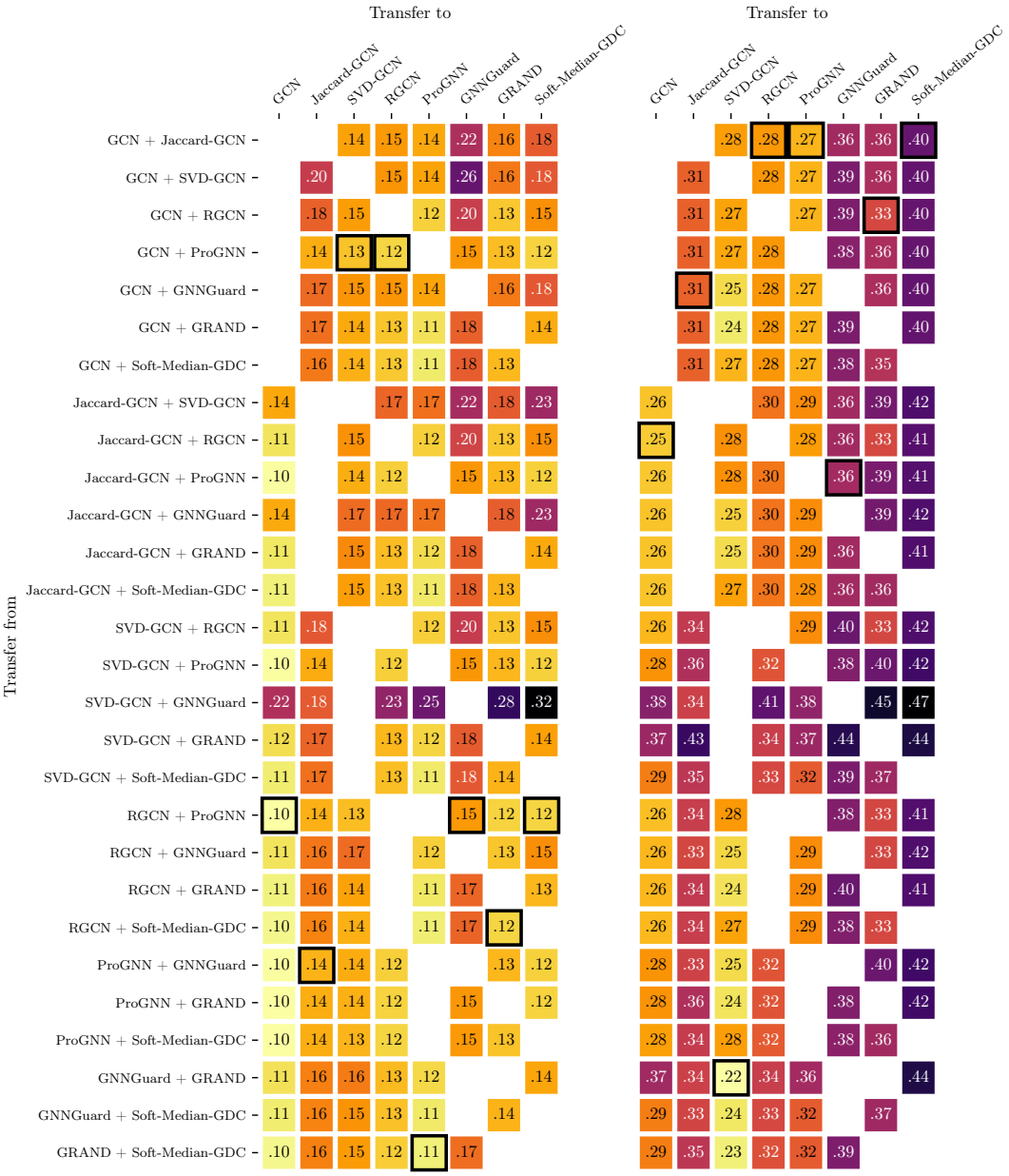
(d) Citeseer, Evasion

Figure F.3: Absolute variant of Fig. 5, showing relative budgets up to 200%.

G Ensemble transferability study

In Fig. 7, we transfer attacks found on an *individual* model to other models. It is natural to also assess the strength of transfer attacks supplied by *ensembles* of models. In Fig. G.1, we address this question for 2-ensembles. For poisoning, the combination of RGCN and ProGNN turns out to be (nearly) the strongest in all cases, which is reasonable since both already form strong individual transfer attacks as is evident in Fig. 7. For evasion, the differences are more subtle.

We also investigate 3-ensembles, but omit the plots due to their size. For poisoning, RGCN and ProGNN now combined with Soft-Median-GDC remain the strongest transfer source, yet the improvement over the 2-ensemble is marginal. For evasion, there is still no clear winner.



(a) Poisoning (b) Evasion

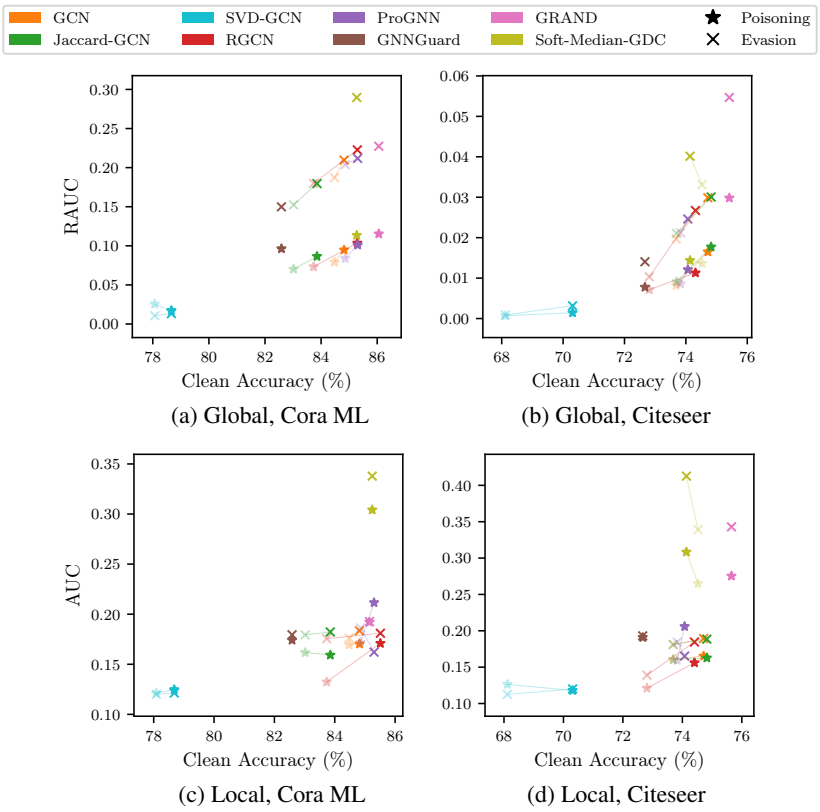
Figure G.1: Variant of Fig. 7 with ensembles of models as attack transfer sources. The color maps are *not* matched across (a) and (b) for improved readability.

H GCN and defense hyperparameters: original vs. tuned for adaptive attacks

To allow for the fairest comparison possible, we tuned the hyperparameters for each model (including GCN) towards maximizing both clean accuracy and adversarial robustness on a single random data split. In Table H.1, we list all hyperparameter configurations. While we cannot run an exhaustive search over all hyperparameter settings, we report substantial gains for most defenses and the GCN in Fig. H.1. The only exceptions are GRAND, Soft-Median-GDC on Cora ML, and GNNGuard. For GRAND, we do not report results for the default hyperparameters as they did not yield satisfactory clean accuracy. Moreover, for Soft-Median-GDC on Cora ML and GNNGuard we were not able to substantially improve over the default hyperparameters.

For the GCN, tuning is important to ensure that we have a fair and equally-well tuned baseline. A GCN is the natural baseline since most defense methods propose slight modifications of a GCN or additional steps to improve the robustness. For the defenses, tuning is vital since they were most originally tuned w.r.t. non-adaptive attacks. In any case, the tuning should counterbalance slight variations in the setup.

As stated in the introduction, each attack only provides an upper bound for the actual adversarial robustness of a model (with fixed hyperparameters). A future attack of increased efficacy might lead to a tighter estimate. Thus, when we empirically compare the defenses to a GCN, we only compare upper bounds of the respective actual robustness. However, we attack the GCN with state-of-the-art approaches that were developed by multiple researchers specifically for a GCN. Even though we also tune the parameters of the adaptive attacks, we argue that the robustness estimate for a GCN is likely tighter than our robustness estimate for the defenses. In summary, the tuning of hyperparameters is necessary that we can fairly compare the robustness of multiple models, even though, we only compare upper bounds of the true robustness.



(a) Global, Cora ML (b) Global, Citeseer

(c) Local, Cora ML (d) Local, Citeseer

Figure H.1: Each defense's clean accuracy vs. (R)AUC values of the strongest attacks, akin to Fig. 6. Muted (semi-transparent) colors represent untuned defenses (except for Soft-Median-GDC on Cora ML and GNNGuard), solid colors denote tuned defenses, and lines connect the two. Our tuned defenses are almost always better than untuned variants w.r.t. both clean accuracy and robustness.

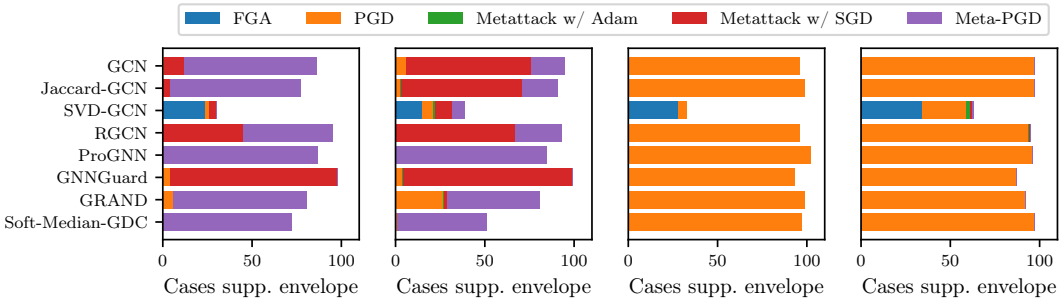
Table H.1: GCN and defense hyperparameters.

	Tuned		Hidden	Dropout				Max epochs	Patience	LR	L_2 reg.						
GCN	×		1×16	0.5				3000	50	0.01	0.0005						
	✓		1×64	0.9				3000	50	0.01	0.001						
Jaccard-GCN	Tuned		Hidden	Dropout	ϵ			Max epochs	Patience	LR	L_2 reg.						
	×		1×16	0.5	0.0			3000	200	0.01	0.0005						
	✓		1×64	0.9	0.0			3000	50	0.01	0.001						
SVD-GCN	Tuned		Hidden	Dropout	Rank			Max epochs	Patience	LR	L_2 reg.						
	×		1×16	0.5	50			3000	200	0.01	0.0005						
	✓		1×64	0.9	50			3000	50	0.01	0.001						
RGCN	Tuned		Hidden	Dropout	ϵ	γ		Max epochs	Patience	LR	L_2 reg.	β					
	×		1×16	0.6	1e-8	1.0		3000	50	0.01	0.0005	0.0005					
	✓		1×32	0.6	1e-8	1.0		3000	50	0.01	0.01	0.0005					
ProGNN	Tuned		Hidden	Dropout				Max epochs	Patience	LR	L_2 reg.	τ	η	μ	β_1	β_2	β_3	β_4
	×	Cora ML	1×16	0.5				3000	50	0.01	0.0005	2	0.01	0.9	1.0	0.001	1.5	0.0005
		Citeseer	1×16	0.5				3000	50	0.01	0.0005	2	0.01	0.9	1.0	0.0001	1.5	0.0005
	✓	Cora ML	1×16	0.5				3000	50	0.01	0.0005	2	0.01	0.9	1.0	0.1	10.0	0.1
		Citeseer	1×16	0.5				3000	50	0.01	0.0005	2	0.01	0.9	1.0	0.2	20.0	0.2
GNNGuard	Tuned		Hidden	Dropout	Pruning	ϵ		Max epochs	Patience	LR	L_2 reg.						
	×		1×16	0.5	×	1e-6		81	n/a	0.01	0.0005						
GRAND	Tuned		Hidden	Dropout	$\overline{\mathbf{X}}$ dropout	δ	K	Max epochs	Patience	LR	L_2 reg.	S	β	T			
	✓	Cora ML	1×32	0.5	0.5	0.5	8	3000	50	0.05	0.0001	4	1.0	0.5			
		Citeseer	1×32	0.2	0.0	0.5	2	3000	50	0.05	0.0005	2	0.7	0.3			
Soft-Median-GDC	Tuned		Hidden	Dropout	k	α	T	Max epochs	Patience	LR	L_2 reg.						
	×		1×64	0.5	64	0.15	0.5	3000	50	0.01	0.001						
	✓	Citeseer	1×64	0.5	64	0.25	0.5	3000	50	0.01	0.001						

I Comparison of success of attack approaches

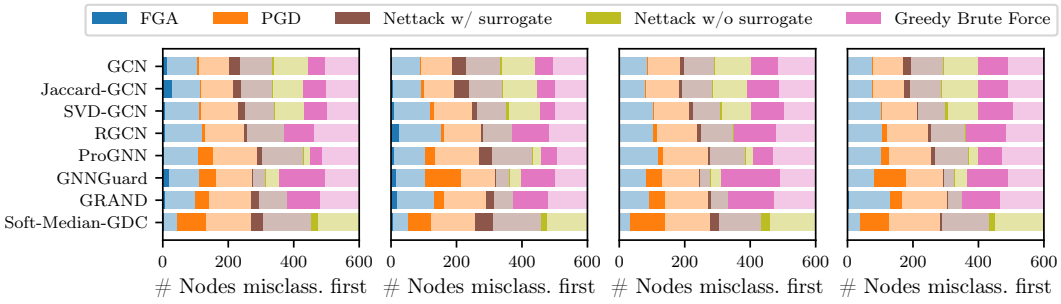
In Fig. I.1 we report which of the global attack techniques generate the strongest attacks, and in Fig. I.3, we break down every global attack attempt. Analogously, in Fig. I.2 and Fig. I.4, we report which local attack techniques require the smallest budget to misclassify the target nodes. In Fig. I.3, we additionally compare different loss types for global attacks.

In general, we can say that PGD is the dominating attack for global evasion. For poisoning, Meta-PGD seems to be the strongest – slightly more successful than Metattack, though not in every case. Greedy brute force dominates the local attacks, but for some defenses, PGD and Nettack have an edge.



(a) Cora ML, Pois. (b) Citeseer, Pois. (c) Cora ML, Evas. (d) Citeseer, Evas.

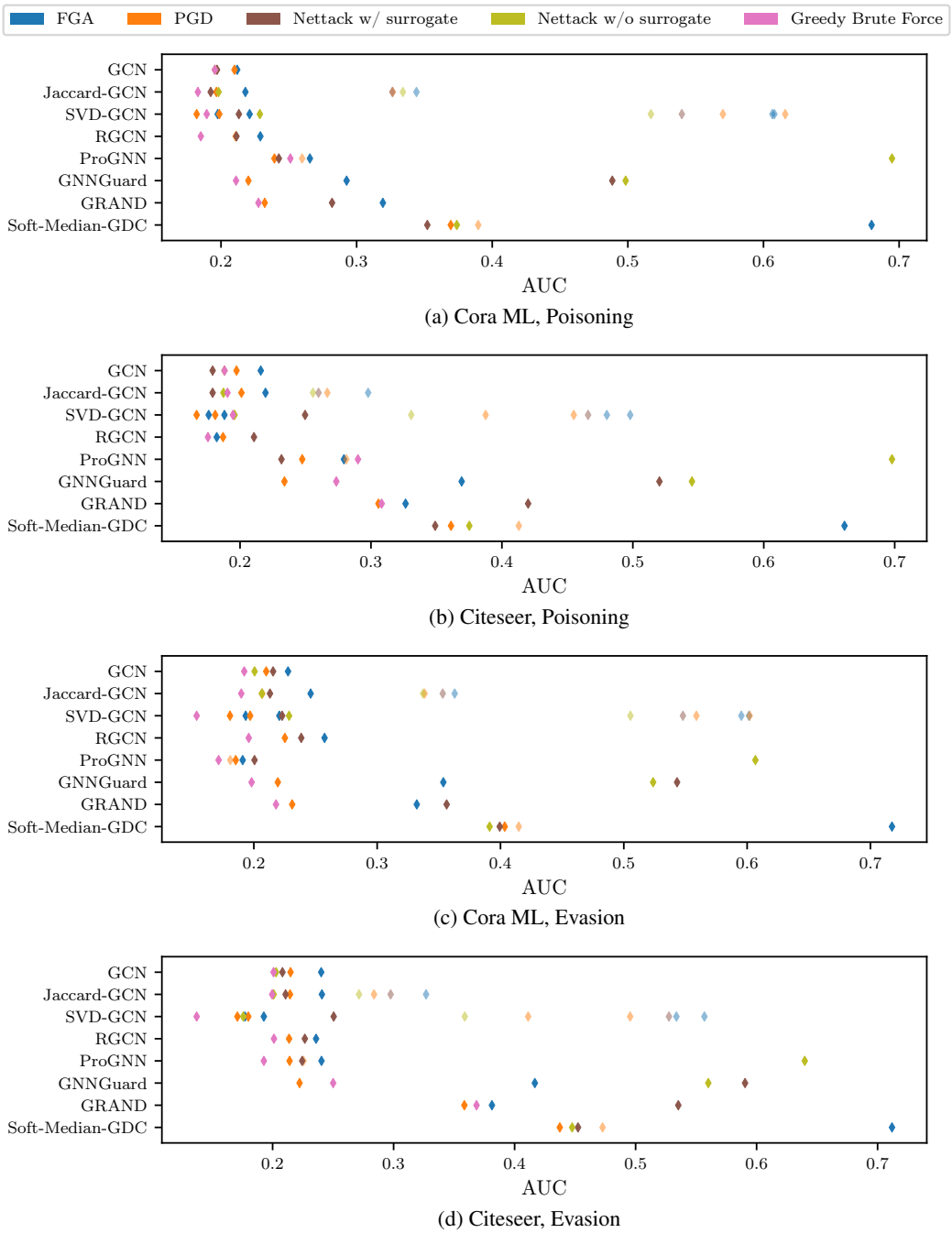
Figure I.1: Number of global attack attempts which support the envelope curve over all attack attempts, as introduced in Fig. 3. We observe that for evasion, PGD almost always yields the strongest attack, while for poisoning, either Metattack, Meta-PGD, or both dominate. Using Adam instead of SGD to train the defense nearly always worsens Metattack's performance.



(a) Cora ML, Pois. (b) Citeseer, Pois. (c) Cora ML, Evas. (d) Citeseer, Evas.

Figure I.2: Number of target nodes for which the respective local attack needs the least budget (among all attacks) to misclassify them. When multiple attacks achieve the same lowest budget, the target node is counted in parts towards each winning attack and drawn with a muted color. We observe that greedy brute force is often the strongest attack; only sometimes, PGD and Nettack beat it on some defenses, especially for poisoning. Using the defense's weights instead of a surrogate model for Nettack is rarely an improvement. Still, for the majority of target nodes, multiple attacks are equally strong in terms of achieving the same lowest budget (tie). We do not run the greedy brute force attack on Soft-Median-GDC due to the costly PPR calculation.

Figure I.3: The RAUC of every global attack we have conducted. Attacks are color-coded by principal technique, and markers indicate the attack loss. Muted colors represent attacks *without* edge masking (Jaccard-GCN), our edge weighting trick (SVD-GCN), multiple PGD auxiliary models (ProGNN), Meta-PGD initialization from ProGNN and unlimited unrolled epochs (GRAND), and PGD initialization from GCN (Soft-Median-GDC). We observe that (1) the TLM and PM losses are superior in almost all cases; (2) PGD attacks are best for evasion while Metattack and Meta-PGD are unsuited; (3) Metattack with SGD and Meta-PGD are best for poisoning while Metattack w/ Adam even falls behind the surprisingly strong evasion-poisoning transfer; (4) FGA is weak for each defense apart from SVD-GCN; (5) the cited adaptions are beneficial as attacks with muted colors are worse; (6) a strong adaptive attack is necessary to reach a low RAUC.



(a) Cora ML, Poisoning

(b) Citeseer, Poisoning

(c) Cora ML, Evasion

(d) Citeseer, Evasion

Figure I.4: The AUC of every local attack we have conducted. Attacks are color-coded by principal technique. Muted colors have the same signification as in Fig. I.3. We observe that (1) greedy brute force is often the best attack, closely followed by PGD, while FGA is not as strong; (2) Nettack can rarely be made stronger by utilizing the target model's weights instead of a surrogate model (red); (3) many defenses successfully defend against Nettack; (4) against those defenses for which we have adapted Nettack, it becomes much stronger (muted vs. normal green); (5) the adaptions are also beneficial for other attacks, as those with muted colors are worse.

J Sensitivity to random seed

When transferring perturbations from evasion to poisoning, a different random seed is used for training the poisoned model than was used for the evasion one. In Fig. J.1, we study using the example of GCN and ProGNN whether poisoning success improves when we instead assume the same seed is used. This is indeed the case and turns out particularly strong on tuned ProGNN. However, by using multiple auxiliary models during evasion as detailed in § 4 under the ProGNN example subheading, we can substantially reduce the dependence of the attack upon a particular random seed and thereby improve attack performance.

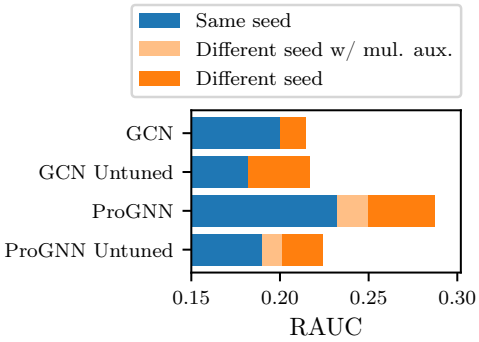


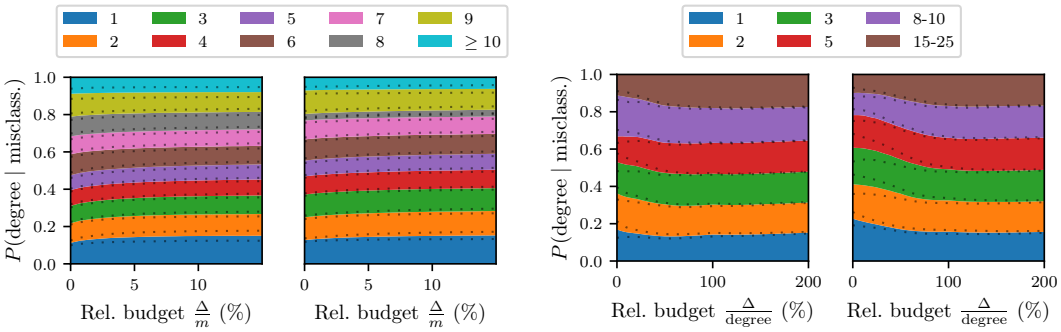
Figure J.1: Lowest RAUC achieved by global evasion-poisoning transfer attacks on Cora ML under the premise that the random seed used by the victim is known respectively unknown to the attacker. While not knowing the seed is disadvantageous especially on ProGNN, our attack using multiple auxiliary models successfully compensates this issue.

K Robustness over node degree

We explore the behavior of nodes under attack depending on their degree. In Fig. K.1, we show the probability that a successfully misclassified node falls into a certain degree range, broken down by relativ budget.

We cannot confirm the prevalent conjecture that global attacks tend to target low-degree nodes, as they are easier to break. Our results show that all degree groups are misclassified uniformly over all budgets. There is no clear preference for lower-degree nodes.

For local attacks, on the other hand, we indeed observe that the success rate of changing the predicted class is independent of the node degree if and only if using a relative budget. For example, when allowing a certain relative budget, e.g., 100% of the target node's degree, we manage to misclassify the same fraction of 1-degree target nodes (with absolute budget of 1) as 5-degree ones (with absolute budget of 5).



(a) Global, Cora ML (b) Global, Citeseer (c) Local, Cora ML (d) Local, Citeseer

Figure K.1: The probability that a misclassified node is in a certain degree range. More specifically, for global attacks, that is which ratios of test set nodes from subsets with degree 1, 2, 3, ... , 9, ≥ 10 are misclassified per budget, normalized s.t. the stacked results sum to 1 everywhere. For local attacks, we show the amount of nodes from each target node set misclassified per budget, again normalized s.t. the stack sums to 1. Results are averaged over all experiments conducted (including evasion and poisoning) on tuned models. The dotted lines indicate standard deviation. We observe no substantial systematic bias towards the misclassification of low-degree nodes.

L Attack characteristics

Next, we present interesting patterns of the adversarial perturbations for each model/defense. We show the *(1) node degree*, *(2) closeness centrality*, *(3) homophily*, *(4) Jaccard similarity* of node attributes, and *(5) the ratio of removed edges* over the strongest edge perturbations in Fig. L.1. For statistics 1-4, we consider the pairs of nodes that were affected by an adversarial edge flip (i.e., insertion or removal). Here we average over the strongest attack found for each budget (without transferring attacks between defenses). Thus, the values indicate what characteristics are important for strong, adaptive attacks.

(1) Node degree. For global attacks, the degree tends to be lower than the average degree of the dataset as given in Table F.1. The higher average degree for local attacks might be influenced by the node selection. Interestingly, on SVD-GCN attacks connect very high-degree nodes, most likely because high-degree nodes correspond to dimensions represented by the most significant eigenvectors of $\mathbf{A}$ (see § 4 Example 1 and § E.2). The attacks exploit the sensitivity of SVD-GCN to perturbations of high-degree nodes. This could hint towards how adaptive attacks catastrophically break SVD-GCN.

(2) Closeness centrality. The closeness centrality of a particular node v is one over the sum of distances from v to all other nodes in the graph, multiplied by the total number of nodes in the graph. Attacks against SVD-GCN connect very central nodes, which probably correlates with them having high degrees. Interestingly, also the perturbations for GNNGuard seem to be of slightly increased centrality.

(3) Homophily refers here to the fraction of pairs of nodes that share the same class. Successful adaptive attacks on Jaccard-GCN share the same homophily as those on GCN, indicating that the Jaccard coefficient is not suited to filter heterophil edges. Attacks on SVD-GCN, GNNGuard, and Soft-Median-GDC have higher homophily than those on GCN, hinting that these defenses successfully filter some heterogeneous edges, forcing some attacks to adapt.

	GCN	Jaccard-GCN	SVD-GCN	RGCN	ProGNN	GNNGuard	GRAND	Soft-Median-GDC
Global	1.98	2.10	19.9	1.87	2.03	3.18	2.03	2.26
Local	4.26	4.33	19.9	4.26	4.19	5.56	4.35	4.78

(a) Node degree

Global	.126	.129	.142	.126	.127	.134	.128	.131
Local	.131	.131	.151	.132	.131	.139	.133	.134

(b) Closeness centrality

Global	.023	.024	.061	.014	.043	.068	.021	.121
Local	.027	.031	.041	.030	.016	.142	.056	.181

(c) Homophily

Global	.028	.037	.025	.026	.032	.095	.022	.034
Local	.030	.038	.028	.028	.026	.081	.024	.045

(d) Jaccard similarity

Global	.001	.001	.001	.001	.006	.012	.003	.016
Local	.023	.026	.020	.029	.010	.139	.056	.173

(e) Ratio of removed edges

Figure L.1: Various metrics characterizing the nature of the adversarial edges from our strongest attacks, which are those visible in Fig. I.1 and Fig. I.2, as well as the nature of the nodes connected respectively disconnected by them.

(4) Jaccard similarity. As expected, attacks on Jaccard-GCN have to compensate its filter by picking edges with nonzero coefficient. Attacks against GNNGuard connect nodes with very similar features, presumably to get past its cosine distance-based edge weighting. Curiously, attacks against Soft-Median-GDC behave similarly, yet only in the local setting and less pronounced. This is probably necessary to avoid that the new edges are weighted down as outliers by the robust aggregation, which becomes less of an issue when perturbing a large amount of edges in the global setting and thereby shifting what it means to be an outlier. Other defenses and especially GRAND admit connecting nodes as or more dissimilar than is the case on GCN.

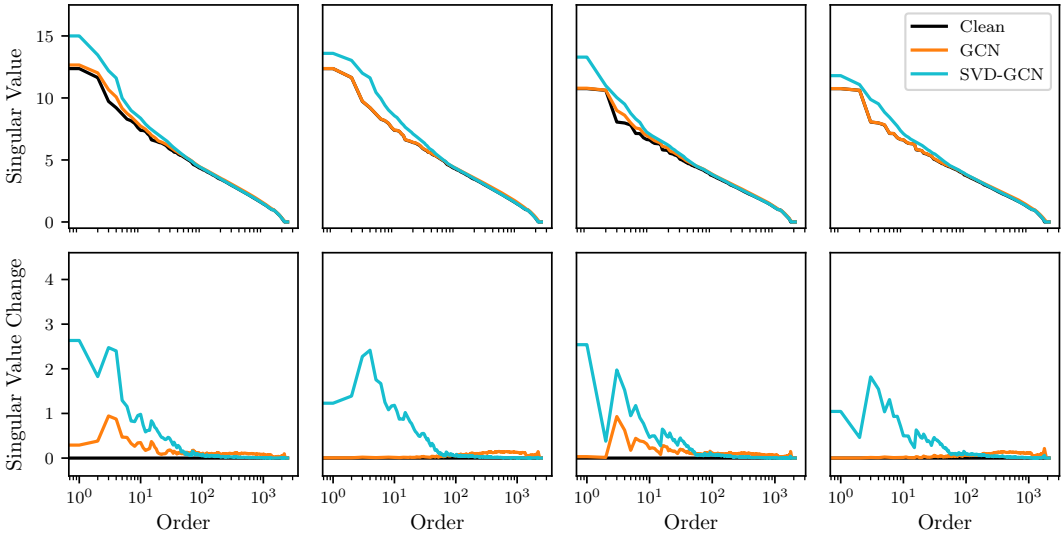
(5) Ratio of removed edges. It is clear to see that for all models, the adversarial attack mostly adds new edges. This indicates that edge insertion is stronger than edge deletion. Strong adaptive attacks on GNNGuard and Soft-Median-GDC seem to require the most edge deletions. Moreover, deletions are of much greater importance for local attacks.

M Spectral properties of adaptive attacks

Previous studies have shown that adversarial attacks tend to focus the high-frequency (i.e., less significant) singular values of the adjacency matrix, both in the local [12] and global [30] setting. In consequence, defenses that exploit this observation to subdue attacks have been proposed (including SVD-GCN and ProGNN). This is a prime example of where (1) defenses were designed to circumvent specific attack characteristics and (2) an intuitive explanation exists of why the defense should improve robustness. However, our adaptive attacks have shown that neither (1) nor (2) entail actual robustness. In the case of SVD-GCN, it seems like the model becomes even less robust. It is only natural to ask whether our attacks exhibit spectral properties different from the high-frequency observation upon which SVD-GCN is built.

In Fig. M.1, we show the spectra of adjacency matrices before and after attacking GCN and SVD-GCN in various settings. Indeed, our adaptive attacks on SVD-GCN perturb more of the low frequencies and less of the high frequencies compared to attacks on GCN. Even though such low frequency-heavy perturbations are hypothesized to be "noticeable" [12, 30], it is unclear how this can be exploited in practice without knowing the clean graph or the underlying distribution of the spectrum. In § A, we give additional reasons why we disregard constraints beyond the L_0 difference.

Fig. M.1 also shows that, in contrast to previous beliefs, effective attacks on a GCN may lie in the low-frequency spectrum (see subplots a and c). This questions the strategy of dampening high-frequency singular values to defend against attacks in the first place.



(a) Cora ML, FGA/PGD (b) Cora ML, Meta (c) Citeseer, FGA/PGD (d) Citeseer, Meta

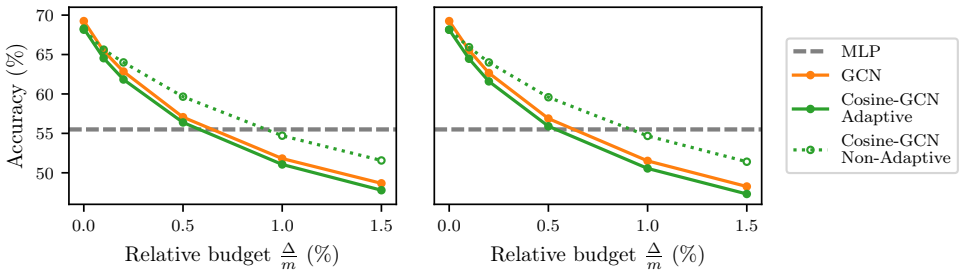
Figure M.1: Singular value spectra of the adjacency matrix before and after perturbation via global adaptive attacks with relative budget of 7.5% against GCN and SVD-GCN. Results are split into native evasion attacks (via FGA and PGD) and native poisoning attacks (via Metattack and Meta-PGD), and averaged in each group. The top row shows the absolute spectrum, and the bottom row the difference to the clean spectrum. The order is plotted logarithmically. We observe that attacks against SVD-GCN strongly perturb the low-order singular values, and it is evident from the relative plots that high-order singular values are perturbed less compared to attacks against GCN.

N On the scalability of adaptive attacks

In our main paper, we do not study adversarial robustness on larger graphs as (a) most defenses do not scale well and (b) we do not want to distract from our finding that structure defense evaluations are overly optimistic. Nevertheless, we consider scalability to be an important aspect for robustness as it is relevant for many applications. As mentioned in § 7, Geisler et al. [17] already study adaptive attacks scaled to large graphs. However, their work is focused on their own defense, and they only consider evasion. For these reasons, we now briefly discuss adaptive attacks on larger graphs.

In Fig. N.1, we show an adaptive attack against "Cosine-GCN" on arXiv from the Open Graph Benchmark [23] (169k nodes). Our Cosine-GCN defense is a natural equivalent of Jaccard-GCN [48] for continuous features. Similarly to Jaccard-GCN on the smaller graphs, Cosine-GCN also comes with some robustness w.r.t. a non-adaptive attack. However, once we apply an adaptive attack, it performs actually slightly worse than the GCN baseline.

Scaling first order attacks. The biggest challenge is certainly that the number of elements in the adjacency matrix scales quadratically with the number of nodes. One way to circumvent this "curse of dimensionality" is to use randomization. For our adaptive attack, we adopt Projected Randomized Block Coordinate Descent (PRBCD) [17]. PRBCD uses the same relaxation as PGD (see § 2 and § A). In each iteration of the attack, it considers only a random subset of edges for gradient update and subsequent projection. Then, for the next iteration, PRBCD keeps edges of high weight and randomly re-samples the edges of low weight. This way, the overhead remains constant in the block size. Since PRBCD is a first-order attack, it is natively adaptive for differentiable models.



(a) Poisoning (b) Evasion

Figure N.1: Adversarial accuracy on the large arXiv dataset per budget for the scalable PRBCD attack against a regular GCN and our Cosine-GCN (single random seed). We use a block size of 1 million edges and run the attack for 200 epochs. Thereafter, we keep the best block for another 50 epochs fixed. Poisoning is conducted by transferring perturbations from evasion.

Evasion vs. poisoning. Gradient-based poisoning attacks seem inherently more challenging since we need to unroll the training. Nevertheless, as long as we can run an evasion attack, there is the possibility to transfer the perturbed adjacency matrix to the poisoning setting. Here, we chose this approach. Still, Zügner & Günnemann [66] show in their appendix that only very few training steps are actually required for Metattack to be effective. Using a low number of training steps is therefore something to consider to scale direct poisoning attacks on larger graphs.