COPYCOP: Ownership Verification for Graph Neural Networks

Rahul Nandakumar
McCombs School of Business
University of Texas at Austin
Austin, TX
rahul.nandakumar@utexas.edu

Deepayan Chakrabarti
McCombs School of Business
University of Texas at Austin
Austin, TX
deepay@utexas.edu

Abstract

Given two GNNs that output node embeddings, how can we determine if they were trained independently? An adversary could have trained one GNN specifically to mimic the other GNN's embeddings. To obscure this relationship between the GNNs, the adversarial GNN might then transform its output embeddings. The two GNNs could have different architectures, weights, and embedding dimensions, and the adversary can transform the embeddings. Despite these stringent conditions, our algorithm (named COPYCOP) can identify such copycat GNNs, unlike existing watermarking and fingerprinting methods. We also provide theoretical guarantees for COPYCOP. Finally, experiments on 14 datasets and 5 GNN architectures demonstrate that COPYCOP is accurate and robust against a broad class of adversarial attacks and transformations. Code is available at: https://anonymous.4open.science/r/CopyCop-Graph-Ownership-Verification-8143/README.md

1 Introduction

Graph Neural Networks (GNNs) are increasingly deployed as reusable embedding models, often exposed through Embeddings-as-a-Service [23]. In this setting, users query a model with graphs and obtain node embeddings that can be reused for downstream tasks. While this enables flexible reuse, it also raises a fundamental question: *can we determine whether another model was independently trained, or derived from a deployed GNN?*

An adversary can train a surrogate GNN to mimic a victim model's embeddings by querying the victim on many graphs and learning from the resulting input-output pairs. The adversary can then apply transformations such as rotation, scaling, permutation, or changes in embedding dimension. These transformations may substantially change the representation while preserving downstream performance, making the surrogate appear unrelated to the victim. Furthermore, the surrogate GNN's architecture can be different from the victim GNN. Thus, given a victim GNN that outputs embeddings, our goal is to identify surrogate GNNs despite changes in architecture, parameters, embedding dimension, or output transformation.

Existing surrogate detection methods fall into two categories: watermarks and fingerprints. Watermarking methods modify training so that the model produces distinctive outputs on special graphs [7, 31, 35]. However, an adversary can bypass such signals through model extraction: by querying the victim on ordinary graphs and training only on the resulting input-output pairs, the surrogate need not reproduce the special watermarked behavior [19].

Fingerprinting methods avoid modifying the victim model and instead rely on intrinsic properties of its outputs. For GNNs that output embeddings, existing methods typically compare the victim and candidate embeddings directly [27]. However, simple transformations such as rotation or changes

Preprint.

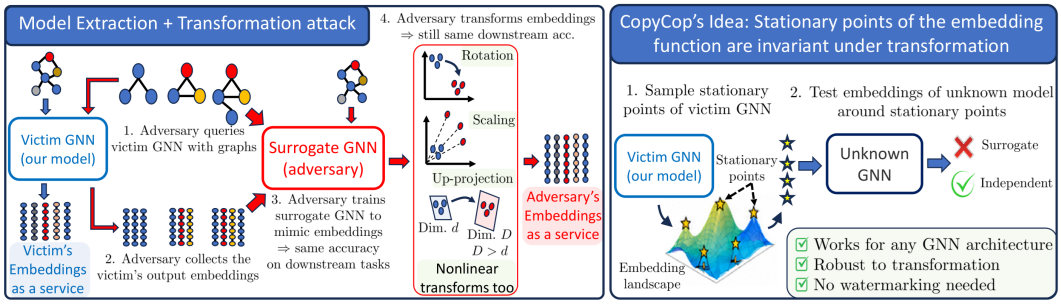


Figure 1: *Overview of* CopyCop*.:* Under the embeddings-as-a-service model, the victim GNN provides embeddings for input graphs, which can then be used for downstream tasks. An adversary can train a surrogate GNN to mimic these embeddings and then transform them, achieving similar accuracy while obscuring the surrogate relationship. CopyCop detects surrogate models under such transformations without watermarking the victim GNN.

in embedding dimension can make embeddings appear different to a classifier while preserving downstream accuracy. As a result, existing embedding-based fingerprints are vulnerable to cosmetic changes in the surrogate representation.

1.1 Our Contributions

We propose CopyCop, a fingerprinting method for GNNs that is robust to a wide range of adversarial transformations, including rotations, scaling, and changes in embedding dimension. Our method is architecture-agnostic: the victim and surrogate GNNs may use different architectures, parameters, and embedding dimensions. Moreover, our fingerprints are randomized and can be regenerated at any time, so leaked fingerprints do not permanently compromise the method. To our knowledge, CopyCop *is the first fingerprinting method for GNN embeddings that works under a broad class of transformations, across GNN architectures, and at any time.*

The key idea is to use stationary points of the embedding function as fingerprints. A node's embedding is a function of the graph structure and node features. Since an adversary may transform the embedding space, the fingerprint must be invariant to such transformations. We prove that stationary points of the embedding function have this invariance property. We show how to sample stationary points from the victim GNN M and test whether these points are also stationary for a candidate GNN Z. This test lets us determine, with high confidence, whether Z is independently trained or a surrogate of M.

We prove that CopyCop detects surrogate models under reasonable conditions, irrespective of GNN architecture or embedding dimension. Empirically, we evaluate CopyCop on 14 datasets and 5 popular GNN architectures. The results show that CopyCop is robust to model extraction, pruning, fine-tuning, and a broad range of embedding transformations.

The rest of the paper is organized as follows. We discuss related work in Section 2. In Section 3, we present our proposed method and prove its robustness against embedding transformations. We empirically validate our approach in Section 4, and conclude in Section 5. All proofs and extra experiments are deferred to the Appendix.

2 Related Work

Our work on ownership verification is related to the broader area of adversarial attacks to infer the graph's structure and properties [10, 29, 33, 34]. Robustness against backdoor attacks is also related to trustworthy AI [6] and integrity verification [12, 16, 28], and query-based schemes for GNNs in MLaaS [30]. Here, we present an overview of the work that is most closely related to us.

Model stealing attacks and defenses: Model stealing attacks and defenses have been studied for images [22] and text [15], among others. One defense is to perturb the model's layers [18], but this fails to counter model extraction attacks [13]. Other defenses detect model extraction attacks based on their query patterns [13]. However, we assume that the victim GNN is public, or the adversarial

queries can be successfully hidden. Similar attacks have been proposed for GNNs [9, 25], but [19] show that current defenses are insufficient. A related but distinct problem is *integrity verification*: detecting whether a deployed model has been tampered with after deployment [12, 16, 28]. However, we focus on *ownership* verification of a separately trained surrogate that may differ entirely in architecture and weights.

Watermarks: Watermarks can be embedded in the victim model's weights by regularizing the model parameters during training [26]. Other approaches embed backdoors, training the model to return improbable outputs for special inputs [2, 8, 17, 36]. Backdoor-based watermarks have also been proposed for GNNs [7, 31, 35]. However, watermarks can degrade performance [27], and are susceptible to model extraction attacks [19].

Fingerprints: Unlike watermarks, fingerprints do not modify the victim model. Fingerprints can be based on model weights [5], special inputs for which some surrogate and independent models give very different outputs [20], pairs of samples along special vectors that straddle decision boundaries [24], or confidence levels of surrogate models [21]. For GNNs, existing methods rely on the similarity of embeddings between victim and surrogate models [27]. This approach applies only when the victim and surrogate GNNs have the same embedding dimension, limiting its generality. In contrast, our approach works for surrogates with any GNN architecture or embedding dimension.

3 Proposed Method

A graph is a tuple (G, X) where $G = (V, E)$ has $n := |V|$ nodes, each with features $\boldsymbol{x}_i \in \mathbb{R}^D$, and $X = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$. Let $\mathcal{D}$ denote the data distribution over feasible pairs (G, X). A d-dimensional GNN H is a function $\boldsymbol{h} : (G, X) \to \mathbb{R}^{d \times n}$, whose i^{th} column $\boldsymbol{h}_i(G, X; H) \in \mathbb{R}^d$ (or $\boldsymbol{h}_i(X)$ when context is clear) is the *embedding* of node i. We are given a d-dimensional *victim* GNN M whose embeddings have the following property.

Assumption 3.1 (Embedding properties)**.** $\boldsymbol{h}_i(X)$ is twice-differentiable with bounded Hessian, and there exist constants $\underline{\alpha}_M$ and $\overline{\alpha}_M$ such that $0 < \underline{\alpha}_M \leq \|\boldsymbol{h}_i\| \leq \overline{\alpha}_M$ for all feasible G, X, and i.

Next, we see a d'-dimensional GNN model Z, possibly using a different architecture and dimension than M. We can query Z with a few graphs and observe its outputs, but we have no access to Z's internals. Our goal is to *determine whether GNN Z is an independently trained model or a surrogate of the victim GNN M*. We define a surrogate model as follows.

Definition 3.2 (Surrogate model)**.** A d'-dimensional GNN M', with embeddings $\boldsymbol{h}'_i := \boldsymbol{h}_i(G, X; M') \in \mathbb{R}^{d'}$, is a surrogate of GNN M if **there exist** functions $\phi : \mathbb{R}^d \to \mathbb{R}^{d'}$ and $\hat{\boldsymbol{h}} : (G, X) \to \mathbb{R}^{d \times n}$ such that

$$\sup_{G,X,i} \|\hat{\boldsymbol{h}}_i(G, X) - \boldsymbol{h}_i(G, X; M)\| \leq \epsilon, \qquad \boldsymbol{h}'_i = \phi\left(\hat{\boldsymbol{h}}_i\right), \tag{1}$$

for every graph (G, X) that is feasible or is in a δ-neighborhood of a feasible graph (ϵ and δ are small positive numbers specified later).

For intuition, the adversary can query the victim model M on graphs (G, X), collect the resulting embeddings $\boldsymbol{h}_i(G, X; M)$ for all nodes, and train a GNN whose embeddings $\hat{\boldsymbol{h}}_i$ approximate $\boldsymbol{h}_i$. The adversary may then apply an additional transformation layer that outputs $\phi(\hat{\boldsymbol{h}}_i)$.

Alternatively, the adversary can train M' end-to-end using (G, X) as input and $\phi(\boldsymbol{h}_i)$ as the target output. Even though the adversary never explicitly constructs an intermediate representation $\hat{\boldsymbol{h}}_i$, it is still useful to characterize the connection between $\boldsymbol{h}_i$ and $\boldsymbol{h}'_i$.

In both cases, the victim and surrogate models output embeddings $\boldsymbol{h}_i \in \mathbb{R}^d$ and $\boldsymbol{h}'_i \in \mathbb{R}^{d'}$, respectively, and our algorithm relies only on these outputs. The link between the two is given by the (unobserved) $\hat{\boldsymbol{h}}_i$ and $\phi(\cdot)$ introduced in Definition 3.2. Their existence is used solely in our analysis.

Remark 3.3. The sup-norm in Equation 1 can be relaxed to the supremum over a subset of graphs (G, X) with a high probability under the data distribution. Furthermore, we only need Assumption 3.1 to hold at around our query tuples, defined later. We keep this form to simplify the exposition.

Next, we motivate our proposed method on the special case of no reconstruction error ($\epsilon = 0$) before discussing the general case.

3.1 The Perfect Reconstruction Case

Here, $\hat{\boldsymbol{h}}_i = \boldsymbol{h}_i$, so M' outputs embeddings $\boldsymbol{h}'_i = \phi(\boldsymbol{h}_i)$. Not all choices of $\phi(\cdot)$ are realistic. For example, $\phi(\boldsymbol{h}) = \mathbf{0}$ would lead to useless embeddings. More generally, if $\phi(\boldsymbol{h}_1) = \phi(\boldsymbol{h}_2)$ for $\boldsymbol{h}_1 \neq \boldsymbol{h}_2$, then $\phi(\cdot)$ loses information. This can hurt performance on downstream tasks, so the adversary will avoid such a $\phi(\cdot)$.

Assumption 3.4 (Local invertibility)**.** The transformation $\phi(\cdot)$ is differentiable and locally invertible:

$$\nabla_{\boldsymbol{w}}\phi|_{\boldsymbol{h}} \;:=\; \lim_{t\to 0}\frac{\phi(\boldsymbol{h}+t\boldsymbol{w})-\phi(\boldsymbol{h})}{t} \;\neq\; \mathbf{0} \qquad \forall \boldsymbol{h}\in\mathbb{R}^d, \boldsymbol{w}\in\mathbb{R}^d, \|\boldsymbol{w}\|=1. \tag{2}$$

Assumption 3.4 ensures that $\phi(\boldsymbol{h}_1) \neq \phi(\boldsymbol{h}_2)$ for any two close embeddings $\boldsymbol{h}_1 \neq \boldsymbol{h}_2$. The assumption holds for many intuitive transformations, such as rotation, non-zero scaling, projection to a higher dimension, and translation, and for any composition of such functions (Lemma A.2 in the Appendix). Note that projecting to a lower dimension loses information, so the adversary will avoid $d' < d$.

We need a fingerprint that is invariant under any transformation satisfying Assumption 3.4. We will show that this strong requirement is met by the stationary points of $\boldsymbol{h}_i(X)$, as defined below.

Definition 3.5. Let F be any GNN. We define the *query tuples* $\mathcal{Q}$, *directional derivative* $\nabla_{\boldsymbol{w}}\boldsymbol{h}_i(G,X;F)$ of an embedding at a query tuple, and the *stationary points* $\mathcal{S}(F)$ of GNN F:

$$\mathcal{Q} = \big\{(G,X,i,\boldsymbol{w}) \mid (G,X)\in \mathrm{supp}(\mathcal{D}),\ i\in\{1,\dots,n\},\ \boldsymbol{w}\in\mathbb{R}^D,\ \|\boldsymbol{w}\|=1\big\}$$

$$\boldsymbol{h}_i^{(\tau\boldsymbol{w})}(G,X;F) = \boldsymbol{h}_i\left(G,\left[\boldsymbol{x}_1,\dots,\boldsymbol{x}_{i-1},\boldsymbol{x}_i+\tau\boldsymbol{w},\boldsymbol{x}_{i+1},\dots,\boldsymbol{x}_n\right];F\right),$$

$$\nabla_{\boldsymbol{w}}\boldsymbol{h}_i(G,X;F) = \lim_{\tau\to 0}\frac{\boldsymbol{h}_i^{(\tau\boldsymbol{w})}(G,X;F)-\boldsymbol{h}_i(G,X;F)}{\tau} \quad \forall (G,X,i,\boldsymbol{w})\in\mathcal{Q}$$

$$\mathcal{S}(F) = \big\{(G,X,i,\boldsymbol{w})\in\mathcal{Q} \mid \|\nabla_{\boldsymbol{w}}\boldsymbol{h}_i(G,X;F)\|=0\big\}.$$

We will refer to $\nabla_{\boldsymbol{w}}\boldsymbol{h}_i(G,X;F)$ as $\nabla_{\boldsymbol{w}}\boldsymbol{h}_i(X)$ when the context is clear.

A stationary point is a graph G with features X such that nudging a node's features $\boldsymbol{x}_i$ in the direction $\pm\boldsymbol{w}$ does not change its embedding $\boldsymbol{h}_i$ to first order. But if $\boldsymbol{h}_i$ is unaffected, so is $\phi(\boldsymbol{h}_i)$, *for any* $\phi(\cdot)$. Thus, this point is also stationary for the surrogate model. The next lemma formalizes this property.

Lemma 3.6. *Let $\boldsymbol{w}\in\mathbb{R}^D, \|\boldsymbol{w}\|=1$, and $\boldsymbol{h}'_i = \phi(\boldsymbol{h}_i)$ for some $\phi(\cdot)$ satisfying Assumption 3.4. Then, $\|\nabla_{\boldsymbol{w}}\boldsymbol{h}_i(X)\| = \mathbf{0}$ if and only if $\|\nabla_{\boldsymbol{w}}\boldsymbol{h}'_i(X)\| = \mathbf{0}$.*

This suggests the following algorithm: given the victim GNN M and the unknown GNN Z, test if $\mathcal{S}(M) = \mathcal{S}(Z)$. By Lemma 3.6, the test succeeds if Z is a surrogate of M. But we expect the test to fail if Z is an independent model, since every such model should converge to a different local optimum. Now, we observe empirically that $\mathcal{S}(M)$ and $\mathcal{S}(Z)$ are large sets, making direct comparison difficult. Hence, we use a sampling-based approach, as follows.

Algorithm 1 DetectSurrogate (Special case)

Input: GNNs M and Z

1: Draw independent samples T from $\mathcal{S}(M)$
2: Check if all points in T are stationary for Z
3: **return** *Surrogate* if $T\subseteq\mathcal{S}(Z)$ else *Independent*

In Section 3.2, we will show how we sample from $\mathcal{S}(M)$. We will also remove the need to compute Z's gradients. Next, we formalize the assumption of differences between the stationary points of the victim and any independent model, and use it to prove the algorithm's correctness.

Assumption 3.7 (Idiosyncratic stationary points)**.** (a) $\mathcal{S}(M)$ is equipped with a probability measure μ_M from which we can draw samples. (b) There exists $\gamma_M > 0$ such that for any independently trained model I, $\mu_M\big(\mathcal{S}(M)\setminus\mathcal{S}(I)\big) \;\geq\; \gamma_M$.

Informally, if we sample a stationary point from $\mathcal{S}(M)$ according to μ_M, then with probability at least γ_M this point is not also a stationary point for the independent model I.

Theorem 3.8. *If $\epsilon = 0$ and Assumptions 3.1, 3.4, and 3.7 hold, then Algorithm 1 is correct with probability at least $1 - e^{-2|T|\gamma_M^2}$.*

3.2 General Case

Now, $\hat{\boldsymbol{h}}_i(X)$ can be any function within a band of width ϵ around $\boldsymbol{h}_i(X)$ (Definition 3.2). Hence, its stationary points may differ from M. The function $\hat{\boldsymbol{h}}_i(X)$ could even be discontinuous, so we cannot assume that its directional derivatives exist. So we cannot directly check for stationary points in Algorithm 1. Instead, we approximate the stationarity checks, as shown below.

Definition 3.9. Let F be any GNN model. For a tuple $t = (G, X, i, \boldsymbol{w}) \in \mathcal{Q}$ and $\delta > 0$, define,

$$q_F(t) = \frac{\|\boldsymbol{h}_i^{(\delta \boldsymbol{w})}(X) - \boldsymbol{h}_i(X)\|}{\|\boldsymbol{h}_i(X)\|}, \qquad \beta_F = \frac{E_{t\sim\mu_M} q_F(t)}{E_{t\sim\mathcal{D}_{exp}} q_F(t)}, \tag{3}$$

where $\boldsymbol{h}_i^{(\delta \boldsymbol{w})}(X)$ depends on t as defined in Definition 3.5,

$$(G, X) \sim \mathcal{D}, \quad i \mid G, X \sim \text{Uniform}(1, \ldots, n), \quad \boldsymbol{w} \mid G, X, i \sim \text{Uniform}(\{v \in \mathbb{R}^D,\, \|v\| = 1\}).$$

For intuition, $q_F(t)$ is a (normalized) magnitude of change in an embedding when we change the features $\boldsymbol{x}_i$ of node i in the direction $\boldsymbol{w}$, and β_F compares the expected change near the stationary points of M versus randomly chosen points in $\mathcal{Q}$. The formula for β_F is also invariant under common transformations of the embeddings (Theorem A.3). We will show below that β_F is small when F is the victim model or a surrogate, but not when F is an independent model. Thus, β_F mimics the properties of the directional derivative at stationary points $\mathcal{S}(M)$.

Lemma 3.10. *Under Assumption 3.1, for δ small enough, $\beta_M = O(\delta)$.*

Lemma 3.10 shows that β_M is small for the victim model. Next, consider $\beta_{M'}$ for a surrogate model M'. Here, we need a stronger version of Assumption 3.4. Instead of just requiring $\|\phi(\boldsymbol{v}_1) - \phi(\boldsymbol{v}_2)\| \neq 0$ when $\|\boldsymbol{v}_1 - \boldsymbol{v}_2\| \neq 0$, we now bound the change in norm due to $\phi(\cdot)$.

Assumption 3.11. $\phi(\mathbf{0}) = \mathbf{0}$, and there exist positive constants c, C such that

$$c\|\boldsymbol{v}_1 - \boldsymbol{v}_2\| \;\leq\; \|\phi(\boldsymbol{v}_1) - \phi(\boldsymbol{v}_2)\| \;\leq\; C\|\boldsymbol{v}_1 - \boldsymbol{v}_2\|. \tag{4}$$

Lemma 3.12. *Suppose Assumptions 3.1 and 3.11 hold, and ϵ is as defined in Definition 3.2. Then, for δ small enough and any surrogate M' with $\epsilon = o(\delta)$,*

$$\beta_{M'} \;=\; O\big((C/c)^2 \cdot \max(\epsilon/\delta,\, \delta)\big)\,. \tag{5}$$

The adversary wants a low reconstruction error ϵ so that the downstream accuracy of the surrogate M' is comparable to that of the victim M. But, by Lemma 3.12, a small ϵ forces M' to have a small $\beta_{M'}$, just like β_M (Lemma 3.10). This result is independent of how the adversary trains M'. Next, consider an independent model I. By Assumption 3.7, at least a γ_M fraction of the stationary points of M are unique to M, so they are not stationary for I. We further qualify this assumption next.

Assumption 3.13. Idiosyncratic stationary points of M are not special for independent models I:

$$E_{t\sim\mu_M}[q_I(t) \mid t \notin \mathcal{S}(I)] \;=\; E_{t\sim\mathcal{D}_{\text{exp}}}[q_I(t)]\,. \tag{6}$$

Assumption 3.13 states that the idiosyncratic stationary points of M are not special for I in any way. We only require the independent model I to deviate significantly from M for these points; however, we keep this version to simplify the exposition. Figure 6 in the Appendix shows some evidence in favor of this assumption. With this assumption, we show that $\beta_I = O(1)$, unlike surrogate models.

Lemma 3.14. *Let M and I be independent models. Suppose Assumptions 3.1 holds for both models, and Assumptions 3.7 and 3.13 hold. Then, for δ small enough, $\beta_I \geq \gamma_M + O(\delta)$, where γ_M is defined in Assumption 3.7.*

Our goal is to determine if the GNN Z is a surrogate of the victim M or an independent model. Lemmas 3.10, 3.12, and 3.14 show that β_Z is small ($O(\delta)$ or $O(\epsilon/\delta + \delta)$) if Z is the victim model or a surrogate, but at least γ_M if Z is independent. This motivates Algorithm 2 (also see Remark A.1 in Appendix A).

Theorem 3.15. *Under the assumptions of Lemmas 3.12 and 3.14 and large enough ℓ, Algorithm 2 with $\theta = \gamma_M/2$ is correct with high probability.*

Algorithm 2 DetectSurrogate (General case)

Input: GNNs M and Z, number of samples ℓ, threshold θ

1: $T \leftarrow \ell$ samples from $\mathcal{S}(M)$ ▷ Def. 3.5
2: $R \leftarrow \ell$ samples from $\mathcal{D}_{\text{exp}}$ over $\mathcal{Q}$ ▷ Def. 3.9
3: $\hat{\beta}_Z \leftarrow \big(\sum_{t\in T} q_Z(t)\big)/\big(\sum_{t\in R} q_Z(t)\big)$ ▷ Eq. 3
4: **return** *Surrogate* if $\hat{\beta}_Z \leq \theta$ else *Independent*

Selecting θ in Algorithm 2: Theorem 3.15 shows that θ depends on the unknown parameter γ_M. However, γ_M is upper-bounded by β_I for any independent model I (Lemma 3.14). Thus, we can train several independent GNNs on the same data as M to get an upper bound for γ_M, which we then use to set the threshold. Alternatively, one can use the distribution of β_I over all datasets and GNN architectures (Figure 6 in the Appendix).

Sampling stationary points: Recall that a stationary point is a tuple $(G, X, i, \boldsymbol{w}) \in \mathcal{Q}$ such that $\|\nabla_{\boldsymbol{w}} \boldsymbol{h}_i(G, x; M)\| = 0$ (Definition 3.5). To find such a point, we sample a point $(G, X_0, i, \boldsymbol{w}) \sim \mathcal{D}_{exp}$ (Definition 3.9) and solve

$$X = \arg\min_X \left(\frac{\|\nabla_{\boldsymbol{w}} \boldsymbol{h}_i(G, X; M)\|}{\|\boldsymbol{h}_i(G, X; M)\|} + \lambda \cdot \frac{\|X - X_0\|_F}{\|X_0\|_F} \right). \tag{7}$$

We then use $(G, X, i, \boldsymbol{w})$ when a sample from $\mathcal{S}(M)$ is needed in Algorithm 2. In the first term, we normalize $\|\nabla_{\boldsymbol{w}} \boldsymbol{h}_i\|$ by $\|\boldsymbol{h}_i\|$ since this matches the form of q_M (Definition 3.9). The second term ensures that the chosen node features X are close to X_0 drawn from the data distribution. Hence, (G, X) looks like a realistic graph. We use a solver that does not require access to gradients [3]. Figure 5 in Appendix B shows that the graphs picked using Equation 7 are all distinct.

The optimization in Eq. 7 operates on the node features X, not on the graph structure G. For very large graphs, it is not necessary to optimize over the entire feature matrix X. Since the goal is to find a stationary point for the embedding of a single node i, it suffices to optimize over the features of node i and its k-hop neighborhood. This reduces the per-iteration cost to a forward pass over a local subgraph of size depending on k, rather than the full graph. In our experiments, the embedding dimension has a larger effect on runtime than graph size.

Extension to integer-valued features: When the features are constrained to be integers, we cannot construct $\boldsymbol{h}_i^{(\delta \boldsymbol{w})}$ (Definition 3.5) for arbitrary $\delta > 0$ and direction $\boldsymbol{w}$. Instead, we set $\delta = 1$ and sample $\boldsymbol{w} \in \mathbb{R}^D$ from the set of axis-aligned unit vectors: $\boldsymbol{w} = \boldsymbol{e}_j$, where $j \sim \text{Uniform}\{1, \ldots, D\}$, and $\boldsymbol{e}_j$ is the standard basis-vector along the j^{th} feature. We also replace $\nabla_{\boldsymbol{w}} \boldsymbol{h}_i$ by $\boldsymbol{h}_i^{(\boldsymbol{w})} - \boldsymbol{h}_i$ in Equation 7 when sampling stationary points. While Theorem 3.15 assumes a small δ, we show empirically that this approach works well in practice.

Robustness against countermeasures: Suppose the adversary can examine the queries sent to their surrogate model. The adversary cannot detect or penalize queries about stationary points for two reasons. First, the regularization in Equation 7 makes those graphs look realistic. Second, recall that we query the surrogate with a graph (G, X). Our chosen (G, X) is stationary only for a random node i and direction $\boldsymbol{w}$, both unknown to the adversary.

The adversary could take a different approach by adding noise to their output embeddings. However, this degrades their downstream performance. Furthermore, the adversary does not know how much noise is needed to go beyond CopyCop 's surrogate detection threshold. This depends on CopyCop's scores for independent models, which the adversary does not have access to. We also show in Section 4 that CopyCop is robust against adversarial attacks such as pruning and fine-tuning, which introduce noise.

Finally, note that Algorithm 2 generates new query graphs each time. Thus, even if some query graphs leak, future surrogate detection remains secure.

Dataset	CopyCop					PreGIP				
	GCN	GIN	GSage	ARMA	MixHop	GCN	GIN	GSage	ARMA	MixHop
Citeseer	1.00	1.00	1.00	1.00	1.00	0.75	0.00	0.90	0.85	0.90
OGBMag	1.00	1.00	1.00	1.00	1.00	1.00	0.30	1.00	1.00	0.95
HIV	1.00	1.00	0.64	0.78	1.00	0.65	1.00	1.00	0.50	1.00
Yelp	1.00	1.00	1.00	0.94	1.00	0.65	1.00	0.55	1.00	0.60
MNIST	1.00	1.00	1.00	0.78	1.00	0.80	0.95	0.55	0.80	0.60
BBBP	1.00	1.00	0.93	1.00	1.00	0.60	0.00	0.70	0.20	0.55
Pubmed	1.00	1.00	0.79	0.83	0.93	0.80	1.00	0.40	0.70	0.90
QM9	1.00	1.00	1.00	0.78	1.00	1.00	1.00	1.00	0.90	1.00
Fin	1.00	0.86	0.86	0.78	1.00	1.00	1.00	1.00	1.00	1.00
Amazon	1.00	0.64	1.00	0.67	1.00	0.80	1.00	0.80	0.80	0.90
Coco	1.00	1.00	0.93	0.94	0.93	1.00	0.25	1.00	0.60	0.30
DBLP	1.00	1.00	0.64	1.00	1.00	0.90	0.00	1.00	0.95	0.90
CIFAR	1.00	1.00	1.00	0.94	1.00	1.00	0.65	0.60	0.30	0.70
Computers	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.35	1.00	1.00
Average	**1.00**	**0.96**	**0.91**	**0.89**	**0.99**	0.85	0.65	0.77	0.76	0.81

Table 1: **AUC of surrogate detection under model extraction attack (higher is better):** Each AUC is computed by classifying all surrogates (across all random seeds and all 5 GNN architectures) against all independent models (again across seeds and architectures). PreGIP's predictions are occasionally worse than random (in red). Averaged over 14 datasets, CopyCop dominates for all GNN architectures. It has a perfect AUC for GCN, and is nearly perfect for MixHop.

4 Experiments

We ran experiments to assess the accuracy of CopyCop and competing methods for the surrogate GNN identification problem. We used 14 datasets from diverse domains and tested 5 popular GNN architectures against various adversarial attacks and embedding transformations.

Datasets: We present results on 14 graph datasets covering molecules, citations, co-purchases, social, and financial networks. Each dataset consists of several graphs with associated node features. For datasets containing only a single graph, we used k-hop subgraphs around randomly chosen nodes. All datasets are available from the *PyG* library, and their details are in Appendix B.

Base GNN Models: We used the GCN [14], GIN [32], GraphSage [11], ARMA [4], and MixHop [1] GNN architectures. For each dataset, we trained all models on two data splits per task (node or graph classification/regression). For each model, we set it as the victim and the others as independent models. The surrogate models are trained via the following adversarial attacks.

Competing methods: We compare our method with the recent watermarking method PreGIP [7] and the fingerprinting method GrOVe [27]. PreGIP's surrogate detection does not directly test embeddings. So, for PreGIP, we score every graph by computing mean-distance(G_a, G_b) over watermark graph pairs (G_a, G_b), and normalizing it by the same formula for randomly-chosen pairs of graphs. We compute surrogate detection AUCs using these scores. Using other formulas, such as cosine similarity, reduces PreGIP's performance (Appendix B).

4.1 Accuracy under Model Extraction Attack

We first show that watermarking via PreGIP is vulnerable to model extraction attacks. Here, the adversary samples graphs, gets their embeddings from the victim model, and trains a GNN to output similar embeddings. For each dataset, we trained independent models using multiple GNN architectures across two data splits. Treating each as a victim, we then trained surrogate models. For PreGIP, we followed the same procedure, but watermarked the victim before training the surrogate.

We evaluated both CopyCop and PreGIP on all surrogate and independent models. Each method assigns a score to determine whether a model is a surrogate or independent. Table 1 reports the AUC of surrogate detection based on these scores.

CopyCop is accurate: CopyCop has **perfect surrogate detection (AUC=1)** on 73% of all combinations of datasets and GNN architectures. The AUC exceeds 0.9 in 83% of the cases. CopyCop's average AUC is highest for GCN (always perfect), MixHop (0.99), and GIN (0.96).

Embedding Transformation	CopyCop					PreGIP					GrOVe				
	GCN	GIN	GSage	ARMA	MixHop	GCN	GIN	GSage	ARMA	MixHop	GCN	GIN	GSage	ARMA	MixHop
Permute	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	4%	4%	15%	0%
Rotate	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	4%	10%	16%	0%
Rotate, scale by 5	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	32%	28%	48%	0%
Project($\mathbb{R}^d \to \mathbb{R}^{5d}$), rotate, scale by 5	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	N/A	N/A	N/A	N/A	N/A
Multiply by $d \times d$ Gaussian mat.	1%	1%	2%	2%	2%	2%	1%	4%	7%	3%	0%	18%	29%	35%	0%
(as above) and scale each entry by $\mathrm{Unif}(1, 10)$	2%	1%	2%	3%	3%	3%	2%	3%	7%	4%	7%	52%	50%	53%	8%
Multiply by $5d \times d$ Gaussian mat.	0%	0%	1%	1%	1%	1%	0%	2%	4%	2%	N/A	N/A	N/A	N/A	N/A
(as above) and scale each entry by $\mathrm{Unif}(1, 10)$	1%	1%	1%	1%	1%	1%	0%	2%	4%	3%	N/A	N/A	N/A	N/A	N/A
$\boldsymbol{h}_{ij} \to \tan^{-1}(\boldsymbol{h}_{ij})$	3%	0%	4%	1%	2%	4%	45%	12%	3%	10%	0%	0%	0%	1%	0%
$\boldsymbol{h}_{ij} \to \exp(\boldsymbol{h}_{ij})$	13%	9%	20%	40%	26%	11%	77%	20%	40%	31%	0%	7%	27%	8%	7%
$\boldsymbol{h}_{ij} \to \mathrm{sigmoid}(\boldsymbol{h}_{ij})$	7%	2%	14%	21%	12%	4%	43%	2%	3%	9%	0%	0%	0%	0%	0%
$\boldsymbol{h}_{ij} \to \sinh(\boldsymbol{h}_{ij})$	9%	1%	4%	16%	7%	9%	77%	14%	30%	34%	0%	6%	12%	7%	4%
$\boldsymbol{h}_{ij} \to \tanh(\boldsymbol{h}_{ij})$	3%	0%	5%	1%	2%	10%	48%	15%	8%	15%	0%	0%	0%	1%	0%
$\boldsymbol{h}_{ij} \to \boldsymbol{h}_{ij}^3$	7%	5%	17%	16%	12%	9%	77%	44%	66%	34%	0%	8%	21%	10%	4%
$\boldsymbol{h}_{ij} \to \boldsymbol{h}_{ij}^5$	7%	5%	21%	21%	16%	16%	77%	54%	72%	44%	4%	8%	25%	14%	7%
Translate each entry by 1	8%	2%	14%	21%	12%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
Translate each entry by $\mathrm{Unif}(1, 5)$	8%	3%	18%	26%	18%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
Translate each entry by $\mathrm{Unif}(1, 10)$	9%	3%	20%	26%	22%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
Normalize to unit L_1 norm	1%	0%	3%	6%	1%	22%	56%	37%	18%	20%	0%	0%	0%	2%	0%
Normalize to unit L_2 norm	1%	0%	3%	6%	2%	22%	57%	38%	15%	19%	0%	0%	1%	7%	0%
Average	**4%**	**1%**	**7%**	**10%**	**6%**	5%	28%	12%	13%	11%	15%	22%	25%	26%	16%

Table 2: **Sensitivity to embedding transformations (lower is better):** We report the fraction of independent models for which the classification changes after transforming the surrogate (averaged over 14 datasets). Lower values indicate greater robustness. PreGIP performs poorly for GIN models and under exponential and power transformations. GrOVe is inapplicable when transformations change the embedding dimension (counted as 100% error). CopyCop remains robust everywhere.

CopyCop outperforms watermarks: Across all GNN architectures, **CopyCop's AUC is 17%-48% better** than PreGIP. For GIN, CopyCop outperforms by 48%. Furthermore, PreGIP's scoring is sometimes worse than random (highlighted in red in Table 1).

Recall that in model extraction, the adversary uses the victim model to create a new training set. This set is unlikely to contain PreGIP's watermark graphs. This limits the effectiveness of watermarking. In contrast, CopyCop uses no watermarks and works well for all architectures.

4.2 Accuracy under Embedding Transformations

Having shown that watermarking (PreGIP) is vulnerable to model extraction, we now examine the robustness of fingerprinting (GrOVe) under embedding transformations. We constructed surrogate models by applying various transformations to their output embeddings, including ones that may violate our assumptions, and evaluated all methods on these modified models.

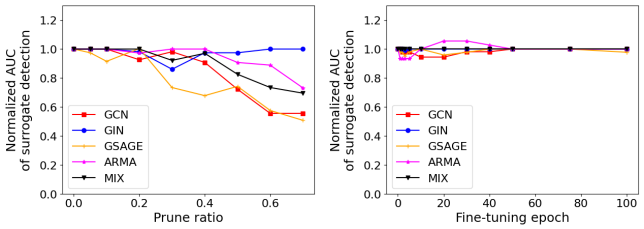
Recall that, given an independent model and a surrogate, we compared their scores to determine which is the surrogate. After transformation, the surrogate's score may change, potentially altering this comparison. To quantify this effect, we measured the fraction of independent models for which the prediction differed before and after transformation. A lower fraction indicates greater robustness. Table 2 reports these fractions, averaged over all 14 datasets.

CopyCop is robust against embedding transformations: CopyCop is immune to rotation, scaling, projection to a higher dimension, and combinations of these (0% change). This matches Theorem A.3. We find that exponentiation has the greatest effect on CopyCop. Exponentiation distorts distances between embeddings (Assumption 3.11), which affects detection accuracy (Lemma 3.12). We note that CopyCop works for translations, even though they do not match Assumption 3.11.

PreGIP and GrOVe are affected in different ways: PreGIP is much more susceptible to exponential and power transformations than CopyCop. In contrast, GrOVe is inapplicable when the surrogate embeddings are projected to higher dimensions. GrOVe is also more sensitive than CopyCop to the scaling of embedding components.

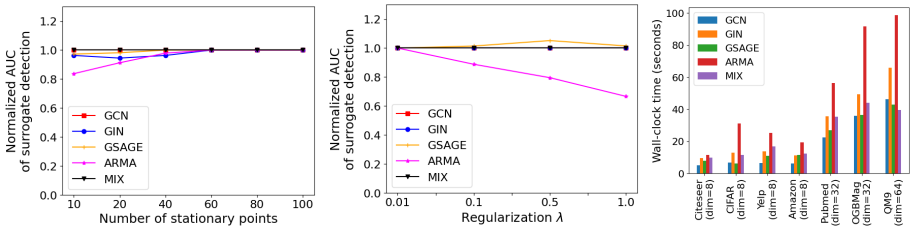
4.3 Accuracy under Other Adversarial Attacks

In a *pruning attack*, the adversary zeros out a fraction of the surrogate model's weights. To evaluate robustness, we compute CopyCop 's AUC for detecting the pruned surrogate among independent models, normalized by the AUC of the unpruned model. Figure 2a reports the trimmed mean of these normalized AUCs across multiple datasets (Citeseer, Pubmed, DBLP, Amazon, Computers).



(a) Effect of pruning (b) Effect of fine-tuning

Figure 2: *AUC of surrogate detection against pruning and fine-tuning attacks:* Results are normalized against no-tuning/fine-tuning.



(a) Varying # of stationary points (b) Effect of varying λ (c) Time to find stat. point

Figure 3: CopyCop needs only 20–40 stationary points to reach near-maximum AUC. AUC is largely insensitive to λ, except for ARMA where $\lambda = 0.01$ is preferred.

CopyCop remains robust up to $\mathbf{40}\%$ pruning, after which the detection AUC declines gradually. Notably, for GIN, the AUC remains stable even with up to 70% of weights pruned.

Next, we consider a *fine-tuning attack*, where the adversary updates the surrogate using a small labeled dataset. Figure 2b shows that CopyCop 's detection AUC remains stable throughout fine-tuning. Meanwhile, the downstream task accuracy saturates after 20–40 iterations (Figure 4 in the Appendix), indicating limited benefit from further updates. Overall, CopyCop is robust to fine-tuning attacks.

4.4 Additional analyses

CopyCop needs only 20–40 stationary points to reach near-maximum AUC across all GNN architectures (Figure 3a), consistent with Theorem 3.8. Surrogate detection is largely insensitive to the regularization λ in Equation 7, except for ARMA where $\lambda = 0.01$ is preferred (Figure 3b). The wall-clock time per stationary point grows with embedding dimension and is shortest for GCN, longest for ARMA (Figure 3c).

5 Conclusions

We introduced CopyCop, a fingerprinting method for verifying the ownership of GNNs that output node embeddings. Our key insight is that stationary points of the embedding function are invariant under a broad class of transformations, making them a robust basis for fingerprinting. This allows CopyCop to detect surrogate models even when the adversary changes architectures, embedding dimensions, or applies complex transformations to the embeddings.

We complement this idea with a practical sampling and testing procedure, along with theoretical guarantees for surrogate detection. Empirically, across 14 datasets and 5 GNN architectures, CopyCop consistently achieves high detection accuracy and outperforms existing watermarking and fingerprinting methods.

One direction of future work is to relax or validate assumptions about the transformation function and the distribution of stationary points. Also, improving the efficiency of stationary point discovery, especially for very large graphs or higher-dimensional embeddings, would further enhance scalability.

References

[1] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. MixHop: Higher-Order Graph Convolutional Architectures via Sparsified Neighborhood Mixing. 2019.

[2] Yossi Adi, Carsten Baum, Benny Pinkas, and Joseph Keshet. Turning Your Weakness Into a Strength: Watermarking Deep Neural Networks by Backdooring. In *SEC'18: Proceedings of the 27th USENIX Conference on Security Symposium*, 2018.

[3] Pauline Bennet, Carola Doerr, Antoine Moreau, Jeremy Rapin, Fabien Teytaud, and Olivier Teytaud. Nevergrad: Black-box optimization platform. *ACM SIGEVOlution*, 14(1):8–15, 2021. ISSN 1931-8499. doi: 10.1145/3460310.3460312. URL `https://doi.org`.

[4] Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Graph neural networks with convolutional arma filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3496–3507, 2022. doi: 10.1109/TPAMI.2021.3054830.

[5] Huili Chen, Bita Darvish Rouhani, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. DeepMarks: A Secure Fingerprinting Framework for Digital Rights Management of Deep Learning Models. In *Proceedings of the 2019 on International Conference on Multimedia Retrieval*, ICMR '19, pages 105–113, New York, NY, USA, June 2019. Association for Computing Machinery. ISBN 978-1-4503-6765-3. doi: 10.1145/3323873.3325042. URL `https://dl.acm.org/doi/10.1145/3323873.3325042`.

[6] Enyan Dai, Tianxiang Zhao, Huaisheng Zhu, Junjie Xu, Zhimeng Guo, Hui Liu, Jiliang Tang, and Suhang Wang. A Comprehensive Survey on Trustworthy Graph Neural Networks: Privacy, Robustness, Fairness, and Explainability. *Machine Intelligence Research*, 21(6):1011–1061, December 2024. ISSN 2731-538X, 2731-5398. doi: 10.1007/s11633-024-1510-8. URL `https://link.springer.com/10.1007/s11633-024-1510-8`.

[7] Enyan Dai, Minhua Lin, and Suhang Wang. PreGIP: Watermarking the Pretraining of Graph Neural Networks for Deep IP Protection. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, pages 415–426, Toronto ON Canada, August 2025. ACM. ISBN 979-8-4007-1454-2. doi: 10.1145/3711896.3737089. URL `https://dl.acm.org/doi/10.1145/3711896.3737089`.

[8] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. DeepSigns: An End-to-End Watermarking Framework for Ownership Protection of Deep Neural Networks. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, pages 485–497, New York, NY, USA, April 2019. Association for Computing Machinery. ISBN 978-1-4503-6240-5. doi: 10.1145/3297858.3304051. URL `https://dl.acm.org/doi/10.1145/3297858.3304051`.

[9] David DeFazio and Arti Ramesh. Adversarial Model Extraction on Graph Neural Networks, December 2019. URL `http://arxiv.org/abs/1912.07721`. arXiv:1912.07721 [cs].

[10] Chenhui Deng, Xiuyu Li, Zhuo Feng, and Zhiru Zhang. GARNET: Reduced-rank topology learning for robust and scalable graph neural networks. In *Learning on Graphs Conference*, 2022. URL `https://openreview.net/forum?id=kvwWjYQtmw`.

[11] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017.

[12] Zecheng He, Tianwei Zhang, and Ruby Lee. Sensitive-sample fingerprinting of deep neural networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4724–4732, 2019. doi: 10.1109/CVPR.2019.00486.

[13] Mika Juuti, Sebastian Szyller, Samuel Marchal, and N. Asokan. PRADA: Protecting Against DNN Model Stealing Attacks. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 512–527, June 2019. doi: 10.1109/EuroSP.2019.00044. URL `https://ieeexplore.ieee.org/document/8806737`.

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

[15] Kalpesh Krishna, Gaurav Singh Tomar, Ankur P Parikh, Nicolas Papernot, and Mohit Iyyer. Thieves on Sesame Street! Model Extraction of BERT-based APIs. 2020.

[16] Deepthi Praveenlal Kuttichira, Sunil Gupta, Dang Nguyen, Santu Rana, and Svetha Venkatesh. Verification of integrity of deployed deep learning models using bayesian optimization. *Knowledge-based systems*, 241:108238, 2022.

[17] Hyun Kwon and Yongchul Kim. BlindNet backdoor: Attack on deep neural network using blind watermark. *Multimedia Tools Appl.*, 81(5):6217–6234, February 2022. ISSN 1380-7501. doi: 10.1007/s11042-021-11135-0. URL https://doi.org/10.1007/s11042-021-11135-0.

[18] Taesung Lee, Benjamin Edwards, Ian Molloy, and Dong Su. Defending Against Neural Network Model Stealing Attacks Using Deceptive Perturbations. In *2019 IEEE Security and Privacy Workshops (SPW)*, pages 43–49, May 2019. doi: 10.1109/SPW.2019.00020. URL https://ieeexplore.ieee.org/document/8844598.

[19] Nils Lukas, Edward Jiang, Xinda Li, and Florian Kerschbaum. SoK: How Robust is Image Classification Deep Neural Network Watermarking? (Extended Version), August 2021. URL http://arxiv.org/abs/2108.04974. arXiv:2108.04974 [cs].

[20] Nils Lukas, Yuxuan Zhang, and Florian Kerschbaum. Deep Neural Network Fingerprinting by Conferrable Adversarial Examples. 2021. arXiv:1912.00888 [cs].

[21] Pratyush Maini, Mohammad Yaghini, and Nicolas Papernot. Dataset Inference: Ownership Resolution in Machine Learning. 2021.

[22] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical Black-Box Attacks against Machine Learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, pages 506–519, New York, NY, USA, April 2017. Association for Computing Machinery. ISBN 978-1-4503-4944-4. doi: 10.1145/3052973.3053009. URL https://dl.acm.org/doi/10.1145/3052973.3053009.

[23] Wenjun Peng, Jingwei Yi, Fangzhao Wu, Shangxi Wu, Bin Bin Zhu, Lingjuan Lyu, Binxing Jiao, Tong Xu, Guangzhong Sun, and Xing Xie. Are You Copying My Model? Protecting the Copyright of Large Language Models for EaaS via Backdoor Watermark. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7653–7668, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.423. URL https://aclanthology.org/2023.acl-long.423/.

[24] Zirui Peng, Shaofeng Li, Guoxing Chen, Cheng Zhang, Haojin Zhu, and Minhui Xue. Fingerprinting Deep Neural Networks Globally via Universal Adversarial Perturbations. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13420–13429, New Orleans, LA, USA, June 2022. IEEE. ISBN 978-1-6654-6946-3. doi: 10.1109/CVPR52688.2022.01307. URL https://ieeexplore.ieee.org/document/9879052/.

[25] Yun Shen, Xinlei He, Yufei Han, and Yang Zhang. Model Stealing Attacks Against Inductive Graph Neural Networks. pages 1175–1192. IEEE Computer Society, May 2022. ISBN 978-1-6654-1316-9. doi: 10.1109/SP46214.2022.9833607. URL https://www.computer.org/csdl/proceedings-article/sp/2022/131600b031/1FlQwoy12IU.

[26] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. Embedding Watermarks into Deep Neural Networks. In *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval*, ICMR '17, pages 269–277, New York, NY, USA, June 2017. Association for Computing Machinery. ISBN 978-1-4503-4701-3. doi: 10.1145/3078971.3078974. URL https://dl.acm.org/doi/10.1145/3078971.3078974.

[27] Asim Waheed, Vasisht Duddu, and N. Asokan. GrOVe: Ownership Verification of Graph Neural Networks using Embeddings, September 2023. URL http://arxiv.org/abs/2304.08566. arXiv:2304.08566 [cs].

[28] Shuo Wang, Sharif Abuadbba, Sidharth Agarwal, Kristen Moore, Ruoxi Sun, Minhui Xue, Surya Nepal, Seyit Camtepe, and Salil Kanhere. Publiccheck: Public integrity verification for services of run-time deep models. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1348–1365. IEEE, 2023.

[29] Xiuling Wang and Wendy Hui Wang. Group Property Inference Attacks Against Graph Neural Networks. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, pages 2871–2884, New York, NY, USA, November 2022. Association for Computing Machinery. ISBN 978-1-4503-9450-5. doi: 10.1145/3548606.3560662. URL https://dl.acm.org/doi/10.1145/3548606.3560662.

[30] Bang Wu, Xingliang Yuan, Shuo Wang, Qi Li, Minhui Xue, and Shirui Pan. Securing graph neural networks in mlaas: A comprehensive realization of query-based integrity verification. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 2534–2552. IEEE, 2024.

[31] Jing Xu, Stefanos Koffas, Oğuzhan Ersoy, and Stjepan Picek. Watermarking Graph Neural Networks based on Backdoor Attacks. In *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*, pages 1179–1197, July 2023. doi: 10.1109/EuroSP57164.2023.00072. URL https://ieeexplore.ieee.org/document/10190545/.

[32] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations (ICLR)*, 2019.

[33] Xiang Zhang and Marinka Zitnik. GNNGUARD: defending graph neural networks against adversarial attacks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, pages 9263–9275, Red Hook, NY, USA, December 2020. Curran Associates Inc. ISBN 978-1-7138-2954-6. URL https://dl.acm.org/doi/10.5555/3495724.3496501.

[34] Zhikun Zhang, Min Chen, Michael Backes, Yun Shen, and Yang Zhang. Inference Attacks Against Graph Neural Networks. In *31st USENIX Security Symposium*, 2022.

[35] Xiangyu Zhao, Hanzhou Wu, and Xinpeng Zhang. Watermarking Graph Neural Networks by Random Graphs. In *2021 9th International Symposium on Digital Forensics and Security (ISDFS)*, pages 1–6, June 2021. doi: 10.1109/ISDFS52919.2021.9486352. URL https://ieeexplore.ieee.org/document/9486352.

[36] Yunqing Zhao, Tianyu Pang, Chao Du, Xiao Yang, Ngai-Man Cheung, and Min Lin. A Recipe for Watermarking Diffusion Models, October 2023. URL http://arxiv.org/abs/2303.10137. arXiv:2303.10137 [cs].

A Remarks and Proofs

Remark A.1. In Algorithm 2, we can use an alternative formula for $\hat{\beta}_Z$:

$$\begin{aligned} \hat{\beta}_Z &= \text{Mean}\left(\{\text{percentile of } q_Z(t) \text{ in } U\}_{t\in T}\right) \\ \text{where } U &= \{q_Z(t) \mid t \in R\}. \end{aligned} \tag{8}$$

Here, U contains samples of $q_Z(t)$ for $t \sim \mathcal{D}_{exp}$, supported on the positive real line. Now, consider a point $t \in T$. If t is a stationary point for Z, $q_Z(t) \approx 0$, so the percentile of $q_Z(t)$ in U is also small. Otherwise, t is an idiosyncratic stationary point of M. In this situation, we expect $q_Z(t)$ to be distributed like a tuple drawn from $\mathcal{D}_{exp}$, i.e., with a percentile score distributed uniformly between 0 and 100. Thus, $\hat{\beta}_Z$ is small if Z is a surrogate but high if Z is independent. This mirrors the behavior of β_Z (Equation 3) while avoiding division-related instability. We use this variant in our experiments.

Lemma A.2 (Closure under composition)**.** *If $f : \mathbb{R}^d \to \mathbb{R}^{d'}$ and $g : \mathbb{R}^{d''} \to \mathbb{R}^d$ each satisfy Assumption 3.4, then $f \circ g$ also satisfies Assumption 3.4.*

Proof of Lemma A.2. Let w be any unit vector and h any point in the domain of g. By the chain rule,

$$\nabla_w (f \circ g)\big|_h = J_f\big|_{g(h)} \cdot J_g\big|_h w = J_f\big|_{g(h)} \cdot \nabla_w g\big|_h.$$

Since g satisfies Assumption 3.4, $\nabla_w g\big|_h \neq 0$. Let $v := \nabla_w g\big|_h$; then $v \neq 0$, so $v/\|v\|$ is a unit vector. Since f satisfies Assumption 3.4,

$$J_f\big|_{g(h)}\, v = \|v\| \cdot \nabla_{v/\|v\|} f\big|_{g(h)} \neq 0.$$

Hence $\nabla_w (f \circ g)\big|_h \neq 0$, so $f \circ g$ satisfies Assumption 3.4. □

Proof of Lemma 3.6. Let J', J^ϕ, and J be the Jacobians of the functions $\boldsymbol{h}_i'(X)$, $\phi(\boldsymbol{h})$, and $\boldsymbol{h}(X)$ respectively (these exist by Assumptions 3.1 and 3.4). By the chain rule,

$$\begin{aligned}\nabla_{\boldsymbol{w}} \boldsymbol{h}_i'(X) &= J'|_X\, \boldsymbol{w} \\ &= J^\phi\big|_{\boldsymbol{h}_i(X)}\, J|_X\, \boldsymbol{w} = J^\phi\big|_{\boldsymbol{h}_i(X)}\, \nabla_{\boldsymbol{w}} \boldsymbol{h}_i(X),\end{aligned}$$

Hence, $\nabla_{\boldsymbol{w}} \boldsymbol{h}_i(X) = 0 \Rightarrow \nabla_{\boldsymbol{w}} \boldsymbol{h}_i'(X) = 0$. Conversely, suppose $\nabla_{\boldsymbol{w}} \boldsymbol{h}_i'(X) = 0$. Then, $J^\phi\big|_{\boldsymbol{h}_i(X)}\, \boldsymbol{v} = \boldsymbol{0}$, where $\boldsymbol{v} := \nabla_{\boldsymbol{w}} \boldsymbol{h}_i(X)$. If $\boldsymbol{v} \neq \boldsymbol{0}$, then $\boldsymbol{0} = J^\phi\big|_{\boldsymbol{h}_i(X)}\, \boldsymbol{v} = \|\boldsymbol{v}\| \cdot \nabla_{\boldsymbol{v}/\|\boldsymbol{v}\|} \phi\big|_{\boldsymbol{h}_i(X)}$, which is impossible by Assumption 3.4. Hence, $\nabla_{\boldsymbol{w}} \boldsymbol{h}_i(X) = \boldsymbol{v} = \boldsymbol{0}$. □

Proof of Theorem 3.8. By Lemma 3.6, Algorithm 1 fails iff Z is independent but $T \subseteq \mathcal{S}(Z)$. Suppose Z is an independent model. Each $t \in T$ is drawn independently using measure μ_M. We have, $E_{t\sim\mu_M}[\mathbb{1}_{t\in\mathcal{S}(Z)}] = P(t \in \mathcal{S}(Z)) \leq 1 - \gamma_M$ by Assumption 3.7. Algorithm 1 fails with probability

$$P\left(\frac{|T \cap \mathcal{S}(Z)|}{|T|} = 1\right) \leq (1 - \gamma_M)^{|T|} \leq \exp\left(-\gamma_M \cdot T\right).$$

□

Proof of Lemma 3.10. For any feasible tuple $t = (G, X, i, \boldsymbol{w})$,

$$q_M(t) = \frac{\|\boldsymbol{h}_i^{(\delta \boldsymbol{w})}(X) - \boldsymbol{h}_i(X)\|}{\|\boldsymbol{h}_i(X)\|} = \frac{\delta \cdot \|\nabla_{\boldsymbol{w}} \boldsymbol{h}_i\| + O(\delta^2)}{\|\boldsymbol{h}_i\|}.$$

For $t \in \mathcal{S}(M)$, $\|\nabla_{\boldsymbol{w}} \boldsymbol{h}_i\| = 0$ so $q_M(t) = O(\delta^2)$ (since $\|\boldsymbol{h}_i\| \geq \underline{\alpha}_M$ by Assumption 3.1). Now, $u_M := E_{t\sim\mathcal{D}_{exp}} \|\nabla_{\boldsymbol{w}} \boldsymbol{h}_i\| > 0$ because the GNN's outputs must be affected by changing inputs for at least some graphs. So, $E_{t\sim\mathcal{D}_{exp}} \geq \frac{\delta u_M + O(\delta^2)}{\overline{\alpha}_M}$. Hence,

$$\beta_M \leq \frac{O(\delta^2)}{\delta u_M / \overline{\alpha}_M + O(\delta^2)} = O(\delta).$$

□

Theorem A.3. *For any model F, β_F is unchanged under any combination of*

$$\begin{aligned} \textit{rotation: } \boldsymbol{h} &\to O\boldsymbol{h} && (O \textit{ orthonormal}) \\ \textit{scaling: } \boldsymbol{h} &\to c\boldsymbol{h} && (c \neq 0) \\ \textit{up-projection: } \boldsymbol{h} &\to [\boldsymbol{h}^T, \underbrace{0, \ldots, 0}_{\ell \textit{times}}]^T && (\textit{arbitrary } \ell). \end{aligned}$$

Proof of Theorem A.3. $q_F(t)$ is clearly invariant under these transformations, so β_F is too. □

Proof of Lemma 3.12.

$$\begin{aligned}&\|\phi(\hat{\boldsymbol{h}}_i^{(\delta \boldsymbol{w})}) - \phi(\hat{\boldsymbol{h}}_i)\| \\ &\leq C \|\hat{\boldsymbol{h}}_i^{(\delta \boldsymbol{w})} - \hat{\boldsymbol{h}}_i\| \\ &\leq C \left[\|\hat{\boldsymbol{h}}_i^{(\delta \boldsymbol{w})} - \boldsymbol{h}_i^{(\delta \boldsymbol{w})}\| + \|\boldsymbol{h}_i^{(\delta \boldsymbol{w})} - \boldsymbol{h}_i\| + \|\hat{\boldsymbol{h}}_i - \boldsymbol{h}_i\|\right] \\ &\leq C \left[2\epsilon + \delta \|\nabla_{\boldsymbol{w}} \boldsymbol{h}_i\| + O(\delta^2)\right] \\ &\leq C \left[\delta \|\nabla_{\boldsymbol{w}} \boldsymbol{h}_i\| + O(\epsilon + \delta^2)\right].\end{aligned}$$

Also,

$$\|\phi(\hat{\boldsymbol{h}}_i)\| \geq c\|\hat{\boldsymbol{h}}_i\| \geq c(\|\boldsymbol{h}_i\| - \epsilon) \geq c(\underline{\alpha}_M - \epsilon) \geq c\underline{\alpha}_M/2$$

for ϵ small enough, where the first inequality follows from Assumption 3.11. Hence,

$$q_{M'}(t) \leq \frac{C\left[\delta\|\nabla_{\boldsymbol{w}}\boldsymbol{h}_i\| + O(\epsilon+\delta^2)\right]}{c\underline{\alpha}_M/2}.$$

$$\text{Similarly, } q_{M'}(t) \geq \frac{c\left[\delta\|\nabla_{\boldsymbol{w}}\boldsymbol{h}_i\| + O(\epsilon+\delta^2)\right]}{2C\overline{\alpha}_M}.$$

Define $u_M := E_{t\sim\mathcal{D}_{exp}}\|\nabla_{\boldsymbol{w}}\boldsymbol{h}_i\| > 0$ as in the proof of Lemma 3.10. Then,

$$\begin{aligned}
E_{t\sim\mu_M}\left[q_{M'}(t)\right] &\leq \frac{C}{c\underline{\alpha}/2}\cdot O(\epsilon+\delta^2) \\
E_{t\sim\mathcal{D}_{exp}}\left[q_{M'}(t)\right] &\geq \frac{c(\delta\cdot u_M + O(\epsilon+\delta^2))}{2C\overline{\alpha}_M} \\
\text{So, } \beta_{M'} &\leq \left(\frac{C}{c}\right)^2 \cdot O\left(\frac{(\epsilon+\delta^2)}{\delta}\right) \qquad\qquad (\text{since } \epsilon = o(\delta)) \\
&\leq \left(\frac{C}{c}\right)^2 O\left(\max(\epsilon/\delta, \delta)\right).
\end{aligned}$$

□

Proof of Lemma 3.14. As in the proof of Lemma 3.10, we define $u_I := E_{t\sim\mathcal{D}_{exp}}\|\nabla_{\boldsymbol{w}}\boldsymbol{h}_i(G,X;I)\| > 0$ and find that

$$E_{t\sim\mathcal{D}_{exp}}[q_I(t)] \geq \frac{\delta u_I + O(\delta^2)}{\overline{\alpha}_I} = \Theta(\delta). \tag{9}$$

For $t \in \mathcal{S}(I)$, we have

$$q_I(t) \geq \frac{\delta\|\nabla_{\boldsymbol{w}}\boldsymbol{h}_i(G,X;I)\| + O(\delta^2)}{\overline{\alpha}_I} = O(\delta^2). \tag{10}$$

Now,

$$\begin{aligned}
E_{t\sim\mu_M}\left[q_I(t)\right] &= P(t\in\mathcal{S}(I))\cdot E_t\left[q_I(t) \mid t\in\mathcal{S}(I)\right] \\
&\quad + P(t\notin\mathcal{S}(I))\cdot E_t\left[q_I(t) \mid t\notin\mathcal{S}(I)\right] \\
&= P(\mathcal{S}(M)\cap\mathcal{S}(I))\cdot O(\delta^2) \\
&\quad + P(\mathcal{S}(M)\setminus\mathcal{S}(I))\cdot E_{t\sim\mathcal{D}_{exp}}\left[q_I(t)\right] \\
&\geq \gamma_M\cdot E_{t\sim\mathcal{D}_{exp}}\left[q_I(t)\right] + O(\delta^2), \\
\text{So, } \beta_I &\geq \frac{\gamma_M\cdot E_{t\sim\mathcal{D}_{exp}}\left[q_I(t)\right] + O(\delta^2)}{E_{t\sim\mathcal{D}_{exp}}\left[q_I(t)\right]} \\
&\geq \gamma_M + O(\delta),
\end{aligned}$$

where we used Equation 10 and Assumption 3.13 in the second equality, and Equation 9 in the last step. □

Proof of Theorem 3.15. Since $\boldsymbol{h}_i$ is upper- and lower-bounded (Assumption 3.1), so is $q_Z(t)$. Then, the numerator and denominator of $\hat{\beta}_Z$ converge to their counterparts for β_Z by Hoeffding bounds, and the result follows. □

Dataset	CopyCop					PreGIP				
	GCN	**GIN**	**GSage**	**ARMA**	**MixHop**	**GCN**	**GIN**	**GSage**	**ARMA**	**MixHop**
Citeseer	1.00	1.00	1.00	1.00	1.00	0.60	0.40	1.00	0.80	0.95
OGBMag	1.00	1.00	1.00	1.00	1.00	1.00	0.00	0.60	0.95	0.85
HIV	1.00	1.00	0.64	0.78	1.00	0.50	0.70	0.90	0.60	0.80
Yelp	1.00	1.00	1.00	0.94	1.00	0.90	0.00	0.55	1.00	0.90
MNIST	1.00	1.00	1.00	0.78	1.00	1.00	0.00	0.60	0.85	0.15
BBBP	1.00	1.00	0.93	1.00	1.00	0.95	0.10	0.70	0.00	1.00
Pubmed	1.00	1.00	0.79	0.83	0.93	0.80	0.25	0.30	0.70	0.90
QM9	1.00	1.00	1.00	0.78	1.00	0.85	0.00	1.00	0.00	0.60
Fin	1.00	0.86	0.86	0.78	1.00	1.00	1.00	1.00	1.00	1.00
Amazon	1.00	0.64	1.00	0.67	1.00	1.00	0.10	0.75	0.75	1.00
Coco	1.00	1.00	0.93	0.94	0.93	0.80	0.00	0.80	0.50	0.50
DBLP	1.00	1.00	0.64	1.00	1.00	0.30	0.00	0.20	0.80	0.60
CIFAR	1.00	1.00	1.00	0.94	1.00	0.00	0.00	0.50	0.10	0.40
Computers	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.55	1.00	1.00
Average	**1.00**	**0.96**	**0.91**	**0.89**	**0.99**	0.76	0.25	0.68	0.65	0.76

Table 3: *AUC for classifying surrogate versus independent models under model extraction attack:* This is similar to Table 1, except that we use cosine similarity for PreGIP instead of distances. CopyCop is comprehensively better, especially for the GIN architecture.

B Experimental Details

Table 5 shows the characteristics of our 14 datasets. Table 3 compares the surrogate detection AUC of CopyCop and a version of PreGIP where the scores are based on cosine-similarity rather than distances between embeddings of pairs of watermarks graphs. PreGIP with cosine similarity shows worse results than using distances, so we focus on the latter in the main paper (Table 1).

Figure 4 shows the normalized accuracy of the surrogate model on a downstream task after being fine-tuned for that task. We report the trimmed means over 5 datasets. All models stabilize within 20-40 epochs, with the greatest benefits occurring for GraphSage and MixHop. Note that CopyCop's surrogate detection accuracy remains robust throughout fine-tuning (Figure 2b).

Figure 3a shows CopyCop's normalized AUC when we vary the number of stationary points used in Algorithm 2 (parameter named ℓ). We need only $20 - 40$ points to achieve a high surrogate detection accuracy.

Figure 5 shows the distribution of cosine similarities between the node features of two stationary points of the same size. The distribution is mostly concentrated around 0, showing that most stationary points are distinct from each other. Hence, Algorithm 2 picks distinct graphs for its queries.

Figure 6 shows the distribution of CopyCop's score $\hat{\beta}_Z$ for all independent models (Equation 8). Suppose the stationary points of the victim model are not in any way special points for the independent models. Then, we would expect the percentile scores to be spread uniformly between 0 to 100, with a mean around $\hat{\beta}_Z = 50$. We see that for most datasets and GNN architectures, the $\hat{\beta}_Z$ scores are indeed around 50. This provides some evidence for our Assumptions 3.7 and 3.13. Table 4 lists the GNN architectures and configurations used in our experiments. All seeds are fixed for reproducibility. We used standard configurations from the literature and did not tune hyperparameters for highest accuracy; our goal was to evaluate whether CopyCop works across a diverse range of trained models.

Compute resources: All experiments were run on a single workstation with an Intel Core i9-10980XE CPU (18 cores / 36 threads), 256 GB of RAM, and 2× NVIDIA RTX 3090 GPUs (24 GB each). The benchmark spans 14 graph datasets and 5 GNN architectures (GCN, GIN, GraphSAGE, ARMA, MixHop) per dataset, with stolen and independent copies for each base model. All (dataset, architecture) pairs are independent of one another, so we ran them as independent processes. Wall-clock per-stationary-point timings reported in Figure 3c is per-process (not summed across parallel workers).

Architecture	Configuration
MixHop	(MixHopConv + ReLU) $\times 2$ + Linear
ARMA	(ARMAConv + ReLU + Dropout) $\times 3$ + Linear
GraphSAGE	(SAGEConv + ReLU + Dropout) $\times 2$ + SAGEConv
GCN	(GCNConv + ReLU) $\times 3$ + GCNConv
GIN	(GINConv + ReLU) $\times 2$ + GINConv

Table 4: *GNN model configurations:* All GNNs output node embeddings, whose accuracy is tested on downstream tasks. For node-level tasks, a final linear layer produces the output. For graph-level tasks, we append GlobalMeanPooling + Dropout + Linear.

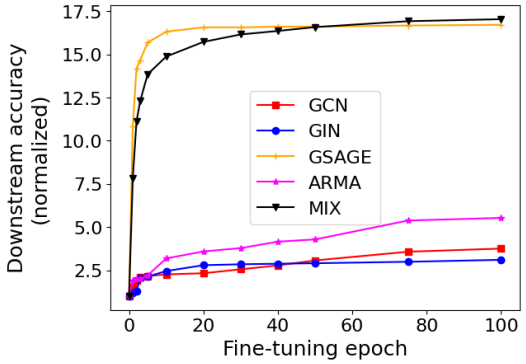


Figure 4: *Accuracy of downstream task under fine-tuning:* The accuracy is normalized by the accuracy before fine-tuning. We report the trimmed mean of the normalized accuracy across 5 datasets. The accuracy stabilizes after 20-40 epochs.

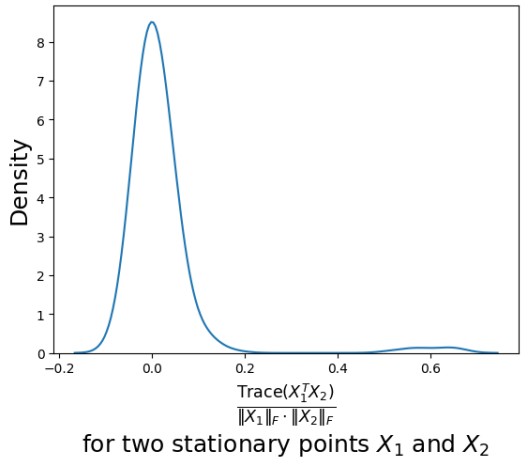


Figure 5: *Cosine similarity of stationary points:* We picked 25 stationary points for 2-node graphs constructed using Equation 7 for the Citeseer dataset. For every pair of graphs, we computed the cosine similarity between their node features after flattening the feature matrices into vectors. We show the distribution of cosine similarity. This shows that most stationary points are nearly orthogonal to each other, showing that our sampling approach in Algorithm 2 picks distinct stationary points.

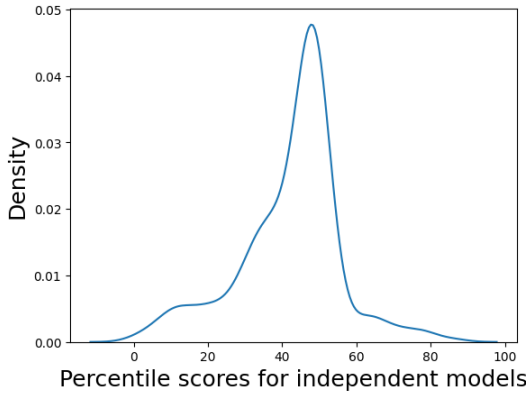


Figure 6: COPYCOP*'s score distribution for independent models:* We show the distribution of $\hat{\beta}_Z$ (Eq. 8) for all independent models across all GNN architectures and datasets. Recall that $\hat{\beta}_Z$ is an average of percentile scores. The distribution peaks around 50%, which is exactly the expected value if the stationary points of the victim model were "random" points for the independent models (see Assumptions 3.7 and 3.13).

Dataset	**#graphs**	**#features**	**embed. dim.**	**integer features?**	**avg. nodes per graph**
Computers	5,000	767	8	Yes	31
QM9	30,000	3	64		10
Amazon	10,000	300	8		83
Coco	30,000	14	8		478
PubMed	8,000	500	32		6
BBBP	800	9	8	Yes	23
HIV	15,000	9	8	Yes	17
OGBMag	8,000	128	32		101
MNIST	5,000	2	8		70
Yelp	5,000	300	8		60
CIFAR	5,000	2	8		118
DBLP	8,000	1,639	8		24
Citeseer	2,000	602	8		8
Fin	4,000	17	8		600

Table 5: *Dataset characteristics*