On Using Embeddings for Ownership Verification of Graph Neural Networks

by

Asim Waheed

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2023

Author's Declaration

This thesis consists of material, all of which I authored or co-authored: see the Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

Asim Waheed is the sole author of Chapter 1, 2, 10, which were written under the supervision of Dr. N. Asokan.

The remainder of this project contains materials and code from the Arxiv report "GrOVe: Ownership Verification of Graph Neural Networks using Embeddings"[1]. Asim Waheed is the first author of this report, Vasisht Duddu and Dr. N. Asokan are co-authors of this report. This thesis is a revised and extended version of that report.

Vasisht came up with the idea of using embeddings as a fingerprint. All authors iterated over and shaped the design of the experiments. Asim setup the experiment pipeline with expert advice from Vasisht, executed the experiments and analyzed the results.

Citation: Waheed, A., Duddu, V., and Asokan, N. (2023). GrOVe: Ownership Verification of Graph Neural Networks using Embeddings. arXiv preprint arXiv:2304.08566.

[1] https://arxiv.org/abs/2304.08566

Abstract

Graph neural networks (GNNs) have emerged as a state-of-the-art approach to model and draw inferences from large scale graph-structured data in various application settings such as social networking. The primary goal of a GNN is to learn an *embedding* for each graph node in a dataset that encodes both the node features and the local graph structure around the node.

Prior work has shown that GNNs are prone to model extraction attacks. Model extraction attacks and defenses have been explored extensively in other non-graph settings. While detecting or preventing model extraction appears to be difficult, deterring them via effective *ownership verification techniques* offers a potential defense. In non-graph settings, *fingerprinting* models, or the data used to build them, have shown to be a promising approach toward ownership verification.

We hypothesize that the embeddings generated by a GNN are useful for fingerprints. Based on this hypothesis, we present GROVE, a state-of-the-art GNN model fingerprinting scheme that, given a *target* model and a *suspect* model, can reliably determine if the suspect model was trained independently of the target model or if it is a *surrogate* of the target model obtained via model extraction. We show that GROVE can distinguish between surrogate and independent models even when the independent model uses the same training dataset and architecture as the original target model.

Using six benchmark datasets and three model architectures, we show that GROVE consistently achieves low false-positive and false-negative rates. We demonstrate that GROVE is *robust* against known fingerprint evasion techniques while remaining computationally *efficient*.

Acknowledgements

I would like to thank my advisor Dr. N. Asokan for providing constant support throughout my program. In his research group he has created a collaborative and healthy working environment that allows ideas to mature while being realistic about timelines and schedules. Under his guidance, I have grown substantially as a researcher and a problem-solver. I would like to thank Dr. Florian Kerschbaum, for it is during his course that the idea for my thesis spawned. And I would like to further thank him and Dr. Urs Hengartner for providing feedback to improve my thesis.

I further thank Vasisht Duddu, for taking out time at ungodly hours of the day to help me with my work, providing technical expertise wherever I got stuck, and Sebastian Szyller for always providing new insight and furthering my knowledge of the field. Lastly, I thank my family and friends for their consistent support and faith in me.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

$\mathcal{A}dv$ Adversary 5, 7–9

$\mathcal{A}dv.\mathcal{A}$ A malicious $\mathcal{A}ccuser$ 9, 10, 37

$\mathcal{A}dv.\mathcal{R}$ A malicious $\mathcal{R}esponder$ 9–11, 24–28, 35

$\mathcal{V}er$ Third-Party Verifier 6, 7, 9–11, 17, 18, 20, 25, 29, 31, 37–40

GROVE Graph Neural Network Ownership Verification 2, 17, 19, 21, 24–32, 35–40

DNN Deep Neural Network 1, 34

FNR False Negative Rate 22, 24–29, 38

FPR False Positive Rate 22, 24, 25, 28

GAT Graph Attention Network 4, 16, 20, 21, 30, 36

GIN Graph Isomorphism Network 5, 20, 21, 30

GNN Graph Neural Network 1–4, 7–9, 11–14, 18, 22, 26, 32–36, 39, 40

GraphSAGE Graph Sample and Aggregate 4, 5, 20, 21, 30

KS test Kolmogorov–Smirnov test 17

ML Machine Learning 1, 3, 4, 7, 17, 32, 36, 39

MLP Multi-Layer Perceptron 5, 7, 21

MSE Mean Squared Error 7, 26

ReLU Rectified Linear Unit 21

Tanh Hyperbolic Tangent Function 21

UAP Universal Adversarial Perturbation 7

List of Symbols

$\mathcal{A}$ Binary adjacency matrix that contains the edges between nodes. 3

$\mathcal{C}$ Classification model. 7, 8

c Cryptographic commitment on a model, such that any subsequent modification to the model also modifies the commitment, e.g. a cryptographic hash function. 10

$c_?$ Cryptographic commitment on the suspect model. 10

c_t Cryptographic commitment on the target model. 10

$\mathcal{F}_{s^2}$ Surrogate model trained via a double extraction attack. 26, 27

h_i Embedding generated by an independent model. 18

$\mathcal{H}_i$ Set of embeddings generated by an independent model. 13, 14

$\mathcal{H}_s$ Set of embeddings generated by a surrogate model. 7, 13, 14, 25

$\mathcal{H}_?$ Set of embeddings generated by a suspect model. 17

$\mathcal{H}_t$ Set of embeddings generated by the target model. 7, 13, 14, 17

h_s Embedding generated by a surrogate model. 18

$\mathcal{H}$ Set of embeddings generated by a GNN. 4, 11, 13

h_t Embedding generated by the target model. 18

$\mathcal{D}_i$ Data used to train the independent model. 5

$\mathcal{F}_i$ Model trained independently from the target model. 5–7, 11, 13, 14, 16–18, 20–22, 24, 25, 30, 36, 39

$\mathcal{D}_v$ Data used to run the input verification process. 10, 14, 18, 20

$\mathcal{C}_{sim}$ Classification model that classifies whether a pair of embeddings are close or far. 17, 18, 21, 24–26, 28–30

$\mathcal{D}_s$ Data used to train the surrogate model. 5, 7, 11, 22

$\mathcal{F}_s$ Model trained via a model extraction attack on a target model. 5–8, 11, 13, 14, 16–18, 20–30, 35, 36, 38–40

$\mathcal{F}_?$ Suspect model that is potentially derived from a target model using a model extraction attack. 6, 7, 9–11, 17, 18, 39

$\mathcal{D}_t$ Data used to train the target model. 5, 6, 10, 11, 22

$\mathcal{F}_t$ Model that is potentially compromised via a model extraction attack. 5–7, 9–11, 13, 14, 16–18, 20–22, 25–27, 30, 35–39

t Secure timestamp on the commitment that can the verifier can verifier. 10

$t_?$ Secure timestamp on the commitment of the suspect model. 10

t_t Secure timestamp on the commitment of the target model. 10

Chapter 1

Introduction

Graph data is ubiquitous and used to model networks such as social networks, chemical compounds, and financial transactions. However, the non-euclidean nature of graph data makes it difficult to analyze using traditional Machine Learning (ML) algorithms. Unlike Euclidean datasets, where the data points are independent, graph data follows homophily, where similar nodes share an edge. To analyze such graph data, a special type of Deep Neural Network (DNN) called GNN has been introduced [30, 19, 81, 68]. These models learn an embedding for each node in the graph that encodes the node features and local graph structure. They can perform node classification [30, 19, 81, 68], link prediction [4, 83, 84, 85], visualization [74], and recommendations [35, 57, 72, 82].

Model builders spend significant time and resources to prepare the training data, perform hyperparameter tuning, and optimize the model to achieve state-of-the-art performance. This makes the deployed GNNs a critical intellectual property for model owners. For instance, Amazon Neptune[1] provides a framework to train and use GNNs; Facebook [34] and Twitter [51] extensively use graph-based ML models. This broad adoption of GNNs makes them vulnerable to model extraction attacks where an adversary tries to train a local *surrogate* model with similar functionality as the deployed *target* model [65, 49, 46, 24, 6, 2].

Model extraction attacks use the input-output pairs from the target model to train the surrogate model. The goal is to achieve similar utility on the primary task for a fraction of the cost. These attacks can potentially violate the company's intellectual property and allow further attacks such as evasion using adversarial inputs and membership inference [49]. While prior work has indicated the feasibility of model extraction attacks

[1] https://aws.amazon.com/neptune/machine-learning/

on various domains [49, 2, 61, 47], recent work introduced such attacks against GNNs as well [10, 77, 55].

Current approaches to address model extraction attacks rely on post-hoc *ownership verification* in which the model owner requests a trusted verifier to decide whether a *suspect model* was stolen from their target model. Ownership verification is done using either fingerprinting or watermarking. Watermarking has been shown to degrade model accuracy [33, 26, 29] and can be evaded [39, 40]. Hence, we identify fingerprinting as a potential scheme that can be used for model ownership verification. *Model fingerprinting* uses inherent features of the model to distinguish between surrogates and independent models. [90, 39, 5, 70, 50, 92]. On the other hand, *dataset fingerprinting* uses the training data as a fingerprint, such that any model trained on the same data as the target model is classified as stolen [40]. Such fingerprinting schemes have been proposed for non-graph DNNs, but there is currently no such work for GNNs.

The state-of-the-art fingerprinting schemes in non-GNN settings use specially crafted inputs to gauge the decision boundary of a model by analyzing the output [40, 39, 50]. The fingerprint, thus, is the set of specially crafted inputs used to query the model. Since GNNs output an embedding for each node, we ask whether the embeddings themselves can be used as a fingerprint. If this works, the benefit of this approach is that the model owner does not generate their own fingerprint. Any data from the same distribution as the model's training data can be used to generate fingerprints.

This thesis presents the **first fingerprinting scheme for GNNs**. We claim the following main contributions:

1. identify GNN embeddings as a potential fingerprint and show that they are useful for verifying model ownership but not dataset ownership (Chapter 4).

2. present GrOVe, embedding-based fingerprinting for GNN model ownership verification (Chapter 5).

3. extensively evaluate GrOVe on six datasets and three architectures (Chapter 7) showing that GrOVe is:

   - effective at distinguishing between surrogate and independent models with close to zero false positives or false negatives (Section 7.1),
   - robust against known fingerprint removal techniques (Section 7.2), and
   - computationally efficient (Section 7.3).

Chapter 2

Background

We describe some preliminaries for Graph Neural Networks (GNNs) and notations used in this thesis (Section 2.1), followed by an overview of model extraction attacks and defenses (Section 2.2).

2.1 Graph Neural Networks

Several real-world applications can be modeled as graphs that include nodes representing different entities in the graph (e.g., authors in citation networks or users in social networks) and edges representing the connections between nodes. Formally, a graph can be represented as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of nodes and $\mathcal{E}$ is a set of edges connecting these nodes. We represent a single node as $v \in \mathcal{V}$ and an edge between nodes u and v as $e_{uv} \in \mathcal{E}$. The entire graph structure, including all the nodes and corresponding edges, can be represented using a binary adjacency matrix $\mathcal{A}$ of size $|\mathcal{V}| \times |\mathcal{V}|$: $\mathcal{A}_{uv} = 1, \forall(e_{uv}) \in \mathcal{E}$.

Machine Learning (ML) on Graphs. Due to the large scale of these graph datasets, ML approaches for analyzing them have gained significant attention. Specifically, GNNs have shown tremendous performance in analyzing graph data for node and edge classification, clustering, and other tasks. Following prior work [55, 10, 77], we consider node classification tasks in this work.

Each node has a feature vector $x \in \mathcal{X}$ and corresponding classification label $y \in \mathcal{Y}$ where $\mathcal{X}$ and $\mathcal{Y}$ are the set of features and labels across all nodes respectively. We can denote the graph dataset as $\mathcal{D} = (\mathcal{A}, \mathcal{X}, \mathcal{Y})$ which is a tuple of the adjacency matrix, the set of node features, and the set of labels respectively.

GNNs are trained to take the graph's adjacency matrix and the features as input and map it to the corresponding classification labels. Once trained, GNNs output a node embedding $h \in \mathcal{H}$. Embeddings are low-dimensional representations of each node and the graph structure. They can be used for downstream tasks such as classification, recommendations, etc. We note that since $\mathcal{H}$ is dependent on the graph structure and the node features, two GNNs trained on different datasets should differ in their output of $\mathcal{H}$ [19, 68]. Formally, the mapping for a GNN is written as: $\mathcal{F} : \mathcal{A} \times \mathcal{X} \rightarrow \mathcal{H}$.

There are two training paradigms for GNNs:

- **Transductive** where the model trains on mapping some graph nodes to classification labels but uses the remaining graph nodes for prediction during testing. Here, the underlying graph structure $\mathcal{A}$, passed as input to the model, remains the same. This is useful for labeling a partially-labeled graph.
- **Inductive** where the model is trained on a training graph dataset but evaluated on an unseen and disjoint testing dataset.

While the transductive setting is useful for analyzing graphs and labeling partially-labeled graphs, it is less useful for ML-as-a-service applications on graph data. Thus, following prior work [55], we consider the more practical setting of inductive training,

GNN Computation. A GNN aggregates information from neighboring nodes to compute the embeddings for a specific node. Formally, each layer of a GNN performs the following operation:

$$h_v^l = \text{AGG}(h_v^{l-1}, \text{MSG}(h_v^{l-1}, h_u^{l-1} : u \in \mathcal{N}(v))) \tag{2.1}$$

$\mathcal{N}(v)$ denotes the nodes that share an edge with v. h_v^l denotes the embedding of node v at layer l. h_v^0 is initialized as the feature vector x for node v. $\text{MSG}(\cdot)$ gathers information from the neighbouring nodes of v, and $\text{AGG}(\cdot)$ aggregates this information with h_v^{l-1} to produce h_v^l. The primary difference between different GNN architectures is the implementation of these two functions.

Graph Sample and Aggregate (GraphSAGE) [19] was the first GNN architecture that uses inductive training. It uses the mean aggregation operation:

$$h_v^l = \text{CONCAT}(h_v^{l-1}, \text{MEAN}(h_u^{l-1} : u \in \mathcal{N}(v))) \tag{2.2}$$

where CONCAT is the concatenation operation, and MEAN is the mean operation.

Graph Attention Network (GAT) [68] includes masked self-attention layers in the GNN, which allows the model to assign a different weight to each neighbor of a node. This

captures the variation in the contribution of different neighboring nodes. The aggregation function is:

$$h_v^l = \text{CONCAT}_{k=1}^{K} \sigma\Big(\sum_{u \in \mathcal{N}(v)} \alpha_{uv}^k \mathbf{W}^k h_u^{l-1} \Big) \tag{2.3}$$

where CONCAT is the concatenation operation, K is the total number of projection heads in the attention mechanism, α_{uv}^k is the attention coefficient in the k^{th} projection head, $\mathbf{W}^k$ is the linear transformation weight matrix, and $\sigma(\cdot)$ is the activation function.

Graph Isomorphism Network (GIN) [81] extends GraphSAGE and uses the aggregation function:

$$h_v^l = MLP^l\big((1+\epsilon^l) \cdot h_v^{l-1} + \sum_{u \in \mathcal{N}(v)} h_u^{l-1}\big) \tag{2.4}$$

where MLP is a multi-layer perceptron and ϵ is a learnable parameter to adjust the weight of node v. This treats $h_u^{l-1} : u \in \mathcal{N}(v)$ as a *multiset*, *i.e.*, a set with possible repeating elements.

2.2 Model Extraction Attacks and Defences

Model extraction attacks consider an Adversary ($\mathcal{A}dv$) who trains a local surrogate model ($\mathcal{F}_s$) to mimic the functionality of a target model ($\mathcal{F}_t$) [65]. These attacks have been extensively studied in many domains including images [46, 49, 24], text [32, 47], and graphs [55, 77, 10] and across different types of models including generative models [61, 23], and large language models [69]. This attack has been identified as a realistic threat that violates the confidentiality of the company's proprietary model and thereby their intellectual property [2].

We denote the training dataset of $\mathcal{F}_t$ (respectively $\mathcal{F}_s$) as $\mathcal{D}_t$ ($\mathcal{D}_s$). Additionally, we refer to models trained independently on a dataset $\mathcal{D}_i$ (in the absence of model extraction attack) are called independent models ($\mathcal{F}_i$).

Non-Graph Model Extraction Attacks. $\mathcal{A}dv$, given query access to $\mathcal{F}_t$, sends queries and obtains corresponding predictions. $\mathcal{A}dv$ then uses these input-prediction pairs to train $\mathcal{F}_s$. Most attacks train $\mathcal{F}_s$ using specially crafted adversarial examples as inputs to $\mathcal{F}_t$ [65, 46, 49, 24]. This helps to ensure that the decision boundary of $\mathcal{F}_s$ is similar to $\mathcal{F}_t$. After a successful attack, $\mathcal{A}dv$ can use $\mathcal{F}_s$ to generate effective transferrable adversarial examples [49] or perform membership inference attacks [56].

Non-Graph Model Extraction Defenses. Preventing model extraction attacks without affecting model performance is difficult [2, 6, 32]. However, ownership verification as a

post-hoc approach helps identify whether a suspect model ($\mathcal{F}_?$) is stolen via model extraction. Normally, this involves a Third-Party Verifier ($\mathcal{V}er$) that verifies ownership. There are currently two main schemes for ownership verification:

- *watermarking* [66, 71, 1, 43, 18, 52, 94, 8, 22, 60] where some secret information is embedded into the model during training which is extracted later during verification.
- *fingerprinting* [39, 40, 50, 5, 90, 92] where inherent features are extracted from a model, without affecting the training process.

Prior work has shown watermarking is brittle and can be easily evaded [38]. Hence, fingerprinting is currently the most promising technique for ownership verification. In this work, we focus on fingerprinting.

Prior work has explored two fingerprinting schemes based on whether the fingerprint is for *dataset ownership* or *model ownership*. The subtle difference between them is how a *different* model trained from scratch *on the same dataset* as $\mathcal{F}_t$ is treated. Dataset-ownership-based fingerprinting classifies such a model as a surrogate. However, model-ownership-based fingerprinting classifies such a model as independent. Here, only models derived (e.g., transfer-learning, model extraction) from $\mathcal{F}_t$ are classified as a surrogate. We describe the main prior works for fingerprinting below.

For dataset ownership-based fingerprinting, **Maini et al.** [40] propose *dataset inference* on the following intuition: the distance of the training data points from the model's decision boundary is increased during training. Hence, the distance of a data point from the decision boundary can help infer its membership in $\mathcal{D}_t$. $\mathcal{V}er$ queries $\mathcal{F}_?$ with data points from $\mathcal{D}_t$ and unseen public data to compute their distances from the decision boundary. $\mathcal{F}_?$ is classified as a surrogate if the distances corresponding to $\mathcal{D}_t$ are large, and the distances corresponding to unseen public data are small.

Most prior work on model ownership-based fingerprinting identifies adversarial examples that can transfer from $\mathcal{F}_t$ to $\mathcal{F}_s$ but not to $\mathcal{F}_i$ [39, 90, 5, 70]. They vary in how to identify such adversarial examples. For instance, data points close to the decision boundary can be used to differentiate between $\mathcal{F}_s$ and $\mathcal{F}_i$ models [5, 70]. Alternatively, untargeted adversarial examples can be used as well [90]. However, these works consider $\mathcal{F}_s$ derived using transfer-learning and fine-tuning but do not consider model extraction attacks [39].

Two prior model ownership-based fingerprinting works are evaluated explicitly with respect to model extraction attacks. We describe them below.

Lukas et al. [39] use an ensemble of models to generate *conferrable adversarial examples*, i.e., adversarial examples that are misclassified by $\mathcal{F}_t$ and $\mathcal{F}_s$, but are not misclassified

by $\mathcal{F}_i$. During verification, $\mathcal{V}er$ computes the difference between the predictions of $\mathcal{F}_t$ and $\mathcal{F}_?$ on the conferrable examples. If the difference exceeds a certain threshold, it is classified as independent, and surrogate otherwise.

Peng et al. [50] use Universal Adversarial Perturbations (UAPs), small imperceptible perturbations which, when added to any image, result in misclassification. They compute a fingerprint by adding the UAP to some data points and compute the change in output before and after adding the UAP. They train an encoder to project the fingerprints into a latent space, such that the fingerprints of $\mathcal{F}_t$ and $\mathcal{F}_s$ are similar but distinct from $\mathcal{F}_i$.

Note that both schemes rely on adversarial examples that cause misclassification. We describe in Section 3.4 why applying these schemes to GNNs is not trivial.

Model Extraction Attacks on GNNs. There is limited literature on model extraction attacks in GNNs. One approach is to use adversarial examples to perform model extraction attacks, similar to the work in non-graph settings [10]. However, the input perturbation for adversarial examples in graph setting is too high to be stealthy. Wu et al. [77] presented seven attacks with different $\mathcal{A}dv$ background knowledge. However, these works are limited to transductive training, impractical in ML as a service setting [55].

This thesis focuses on the more practical case of inductive learning where the training and testing graph datasets are disjointed. Shen et al. [55] presented the first work on model extraction against inductive GNNs. Here, $\mathcal{A}dv$ has access to a query dataset $\mathcal{D}_s = (\mathcal{A}_s, \mathcal{X}_s, \mathcal{Y}_s)$ and the query response (a set of embeddings ($\mathcal{H}_t$) corresponding to the node features ($\mathcal{X}_s$)) it receives from $\mathcal{F}_t$. Using $\mathcal{H}_t$ and the ground-truth labels ($\mathcal{Y}_s$), $\mathcal{A}dv$ trains $\mathcal{F}_s$ to mimic the behavior of $\mathcal{F}_t$. They propose two attacks depending on $\mathcal{A}dv$'s background knowledge: in *Type I* attack, $\mathcal{A}dv$ has access to $\mathcal{X}_s$, $\mathcal{Y}_s$, and the adjacency matrix ($\mathcal{A}_s$) for $\mathcal{D}_s$; in *Type II* attack, $\mathcal{A}dv$ only has access to $\mathcal{X}_s$, $\mathcal{Y}_s$, and uses an edge-estimation algorithm to compute $\mathcal{A}_s$.

In their attack, the architecture for $\mathcal{F}_s$ consists of two components:

- a GNN that takes $\mathcal{X}_s$ and $\mathcal{A}_s$ as input and outputs $\mathcal{H}_s$. While training, this module minimizes the Mean Squared Error (MSE) loss between $\mathcal{H}_s$ and $\mathcal{H}_t$:

$$\mathcal{H}_s = \mathcal{G}nn(\mathcal{X}_s, \mathcal{A}_s) \tag{2.5}$$

$$\mathcal{L}_R = \frac{1}{n_{Ds}} \|\mathcal{H}_s - \mathcal{H}_t\|_{2,1} \tag{2.6}$$

  where n_{Ds} is the number of nodes in $\mathcal{D}_s$.

- an MLP classifier ($\mathcal{C}$) which takes $\mathcal{H}_s$ as input and outputs a class. This is trained to minimize the prediction loss between $\mathcal{Y}_s$ and the predicted labels.

In each epoch, the GNN is first optimized using $\mathcal{L}_R$; then, the parameters are fixed while parameters corresponding to $\mathcal{C}$ are optimized. Both modules are combined to form $\mathcal{F}_s$.

The same training strategy is used to train $\mathcal{F}_s$ using Type I and Type II attacks. However, for Type II attacks, $\mathcal{A}dv$ first estimates $\mathcal{A}_s$. A graph structure is initialized by creating a k-nearest neighbors graph from $\mathcal{X}_s$. This initial graph structure is further optimized using IDGL framework proposed by Chen et. al. [77]. The graph structure is obtained by minimizing a joint loss function that combines prediction loss for node classification and a graph regularization loss that controls the graph's smoothness, connectivity, and sparsity. Additional details about the model extraction attack can be found in [55].

Model Extraction Defenses For GNNs. To the best of our knowledge, there are only two prior works on ownership verification in the context of GNNs [80, 91]. Zhao et al. [91] embed randomly generated subgraphs with random feature vectors as a key. These can be used later to extract the watermark. However, they only focus on node classification tasks. Xu et al. [80] extend the prior work by including graph classification tasks. However, both of these schemes are watermarking schemes. The current literature has no known fingerprinting schemes for GNNs.

Chapter 3

Problem Statement

An effective fingerprinting scheme allows Third-Party Verifier ($\mathcal{V}er$) to identify whether $\mathcal{F}_?$ is a surrogate of $\mathcal{F}_t$ or an independent model. To this end, we outline a system model that defines the interactions between model owners and $\mathcal{V}er$ (Section 3.1), an adversary model describing Adversary ($\mathcal{A}dv$)'s capabilities and goals (Section 3.2), requirements to design an ideal fingerprinting scheme (Section 3.3) and the limitations of prior fingerprinting schemes against Graph Neural Network (GNN) model extraction (Section 3.4).

3.1 System Model

We consider a setting where a proprietary GNN model ($\mathcal{F}_t$) has been developed and deployed as a service. However, $\mathcal{F}_t$ is susceptible to model extraction. An ownership verification system, intended to thwart model extraction, consists of three actors: an $\mathcal{A}ccuser$ (the owner of $\mathcal{F}_t$), a trusted third party $\mathcal{V}er$, and a $\mathcal{R}esponder$ (the owner of a suspect model $\mathcal{F}_?$) who is accused of stealing $\mathcal{F}_t$ by $\mathcal{A}ccuser$. The role of the $\mathcal{V}er$ is to verify whether $\mathcal{F}_?$ was obtained through a model extraction attack on $\mathcal{F}_t$. We refer to a malicious $\mathcal{R}esponder$ as $\mathcal{A}dv.\mathcal{R}$ and a malicious $\mathcal{A}ccuser$ as $\mathcal{A}dv.\mathcal{A}$.

System Design Goals. An ideal system must be robust against both $\mathcal{A}dv.\mathcal{R}$ and $\mathcal{A}dv.\mathcal{A}$.

Case 1 $\mathcal{A}dv.\mathcal{R}$ wants its model $\mathcal{F}_?$, extracted from $\mathcal{A}ccuser$'s $\mathcal{F}_t$, to evade detection.

Case 2 $\mathcal{A}dv.\mathcal{A}$ wants to maliciously claim that $\mathcal{R}esponder$'s $\mathcal{F}_?$ is extracted from $\mathcal{F}_t$.

To address both scenarios, similar to prior work [92, 60], we assume that all model owners are required to securely timestamp their models in a registration step before deployment. This will address $\mathcal{A}dv.\mathcal{A}$ since $\mathcal{A}ccuser$ cannot successfully make an ownership claim against $\mathcal{F}_?$ unless $\mathcal{F}_t$ was registered prior to $\mathcal{F}_?$. In Chapter 9, we explore possible incentives for model owners to register their models. The rest of this thesis focuses on robustness against a $\mathcal{A}dv.\mathcal{R}$.

Model Registration. Model owners are required to:

1. generate a cryptographic commitment c of their model such that any subsequent modification to the model can be detected. One way to compute such a commitment is using a cryptographic hash function.

2. obtain a secure timestamp t on c that $\mathcal{V}er$ can later verify, e.g., by adding c to a blockchain, or utilizing a publicly verifiable timestamping service[1], or receiving a signature from $\mathcal{V}er$ binding c to the current time.

We use subscripts to associate timestamps and commitments to the respective models (e.g., t_t is the secure timestamp on the commitment c_t of $\mathcal{F}_t$)

Dispute Initiation. $\mathcal{A}ccuser$ initiates a dispute by submitting c_t and t_t to $\mathcal{V}er$, and identifying a suspect $\mathcal{R}esponder$. $\mathcal{V}er$ then asks $\mathcal{R}esponder$ to submit $c_?$ and $t_?$ to begin the verification process.

Verification Process. $\mathcal{V}er$ does the following:

1. verifies that t_t, c_t, and $\mathcal{F}_t$ are consistent; and $t_?$, $c_?$, and $\mathcal{F}_?$ are consistent.

2. confirms that $t_t < t_?$. If this condition is not met, the claim is rejected.

3. checks that $\mathcal{F}_t$ and $\mathcal{F}_?$ are well-formed (see below).

4. samples a verification dataset $\mathcal{D}_v$ from the same distribution as $\mathcal{D}_t$.

5. queries $\mathcal{F}_t$ and $\mathcal{F}_?$ with $\mathcal{D}_v$ and passes the outputs to a verification algorithm which decides whether $\mathcal{F}_?$ is a surrogate of $\mathcal{F}_t$ or trained independently.

Step 3 requires $\mathcal{V}er$ to check that a model does not have any non-standard layers. In Chapter 7.2, we explain why this check is necessary.

[1] E.g., https://www.surety.com/digital-copyright-protection

The simplest way for $\mathcal{V}er$ to conduct the checks in the first three steps is for the model owners to send their models ($\mathcal{F}_t$ and $\mathcal{F}_?$) to $\mathcal{V}er$. This may not be feasible for confidentiality or privacy reasons. Thus, the checks can be done either via cryptographic techniques like oblivious inference [36, 27] in conjunction with zero-knowledge proofs [28, 3] or by using hardware-based trusted execution environments [64, 12]. The specific implementation of such protocols is out of the scope of this thesis. For ease of explanation, we limit our discussion to the case where the model owners in a dispute are willing to share their models with $\mathcal{V}er$. Note that it is still necessary to check that the models sent to $\mathcal{V}er$ are indeed the models that were deployed. $\mathcal{V}er$ can do this via a fidelity check [24].

3.2 Adversary Model

$\mathcal{A}dv.\mathcal{R}$'s goal is to train $\mathcal{F}_s$ such that its utility is comparable to $\mathcal{F}_t$. Additionally, $\mathcal{A}dv.\mathcal{R}$ wants high *fidelity* for $\mathcal{F}_s$, i.e., that its inferences match $\mathcal{F}_t$'s. This is useful for mounting subsequent evasion or membership inference attacks against $\mathcal{F}_t$ [49]. Shen et al.'s [55] attack satisfies both these requirements. $\mathcal{A}dv.\mathcal{R}$ may also take additional steps to evade detection.

Attack Setting. Following [55], we consider the *black-box* setting where $\mathcal{A}dv.\mathcal{R}$ has no knowledge of $\mathcal{F}_t$'s hyperparameters or architecture and can only observe $\mathcal{F}_t$'s outputs for given inputs. $\mathcal{A}dv.\mathcal{R}$ has access to a training dataset that is from the same distribution as $\mathcal{F}_t$. As in prior work [55], we assume that the output contains node embeddings ($\mathcal{H}$), useful for downstream tasks such as classification, recommendation engines, visualizations, etc. We assume that $\mathcal{A}dv.\mathcal{R}$ has access to a disjoint non-overlapping dataset $\mathcal{D}_s$ from the same distribution as $\mathcal{F}_t$'s training data $\mathcal{D}_t$. We revisit the details of dataset splits in Chapter 6.

3.3 Requirements for Ownership Verification

We list desiderata for ideal GNN ownership verification schemes that $\mathcal{V}er$ can use to decide if $\mathcal{F}_?$ is stolen from $\mathcal{F}_t$:

R1 Minimize Utility Loss of $\mathcal{F}_t$.

R2 Effective in differentiating between $\mathcal{F}_s$ and $\mathcal{F}_i$ with low false positive/negative rates.

R3 Robust in remaining effective against $\mathcal{A}dv.\mathcal{R}$.

R4 Efficient by imposing a low computational overhead.

3.4 Limitations of Prior Work

We now discuss how the prior works described in Section 2.2, applicable to non-graph and graph datasets, do not satisfy the above requirements.

Non-Graph Datasets. Focusing only on approaches tested against model extraction attacks, we discuss how the three prior fingerprinting approaches for the image domain are not directly applicable to GNNs. We describe the limitations of prior non-graph fingerprinting schemes below.

Maini et al. [40] compute the distance of a data point to the decision boundary by adding noise to the data points, which is not clear for inter-connected graph nodes [10]. Moreover, prior works have indicated that dataset inference incurs false positives [94, 62]. Lukas et al. [39] and Peng et al. [50] rely on adversarial examples as fingerprints. However, unlike images, generating adversarial examples is not trivial for graphs [96, 9]. The related works on generating adversarial examples are described in detail in Chapter 8, but the key takeaway is that the best attack rate achieved by state-of-the-art schemes is only 35%. In the image domain, close to 100% attack rate has been achieved in the current literature.

In summary, adapting non-graph fingerprinting approaches to graph datasets is not trivial, as data records in the image domain are independent. In contrast, nodes in graphs are related and satisfy homophily.

Graph Datasets. Watermarking schemes have been proposed previously for GNNs [91, 80]. However, watermarks in non-graph datasets can be easily removed by model extraction attacks and are hence not robust **R3** [38]. Their effectiveness **R2** against state-of-the-art model extraction attacks is not clear. Finally, watermarks require modifying how the model is trained, violating the non-invasive requirement **R1**. Hence, in this work, we focus on using fingerprints for ownership resolution in GNNs instead of watermarking, which satisfies all the desirable requirements.

Chapter 4

Motivation

We now explore the use of Graph Neural Network (GNN) embeddings as a potential fingerprinting scheme for GNNs against model extraction attacks. We then evaluate whether embeddings are helpful for model fingerprinting or dataset fingerprinting.

Embeddings as Fingerprint for GNNs. Recall from Section 2.1 that two independently trained GNNs (differing in the random seed) should differ in their output of $\mathcal{H}$. Furthermore, the state-of-the-art model extraction attack against GNNs [55] focuses on optimizing *fidelity*, i.e., ensuring alignment in predictions between $\mathcal{F}_s$ and $\mathcal{F}_t$. Hence, $\mathcal{F}_s$ is likely to generate embeddings more similar to $\mathcal{F}_t$ on the same input graph than $\mathcal{F}_i$. This intuition forms the basis of using embeddings as a fingerprint to distinguish between $\mathcal{F}_s$ and $\mathcal{F}_i$ for ownership verification. Note that $\mathcal{H}$ is inherent to GNN model computation. Hence, they do not affect the utility of $\mathcal{F}_t$, satisfying requirement **R1**.

Model Ownership vs. Dataset Ownership. Having identified graph embeddings as a potential fingerprint, we want to verify whether they are helpful for model or dataset ownership. Recall from Section 2.2 that the difference in fingerprinting for model ownership and dataset ownership is in how models trained on the same training dataset are classified. If embeddings generated from two GNNs trained on the same dataset cannot be distinguished, then embeddings are useful as fingerprints for dataset ownership. On the other hand, if embeddings generated from $\mathcal{F}_s$, regardless of the training data, can be distinguished, they are helpful as fingerprints for model ownership. We test this using t-SNE projections of the embeddings to visualize them for different model architectures and datasets [67, 50]. We refer to embeddings generated from $\mathcal{F}_t$, $\mathcal{F}_s$, and $\mathcal{F}_i$ as $\mathcal{H}_t$, $\mathcal{H}_s$, and $\mathcal{H}_i$, respectively.

We describe our experimental setup to infer whether embeddings can be used for data

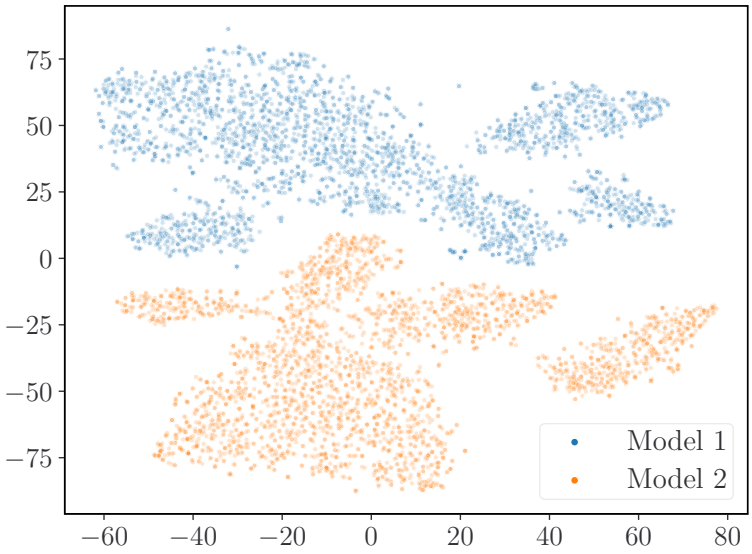
or model ownership verification.

Dataset Ownership. We train two models using either different or the same architecture and training dataset. When the training datasets are different, we split the dataset into three sets: $\mathcal{D}_1$ and $\mathcal{D}_2$, and a verification dataset $\mathcal{D}_v$. We use three different architectures for each model, leading to nine pair-wise combinations. We use six datasets, resulting in 54 model pairs with the same training datasets and 54 model pairs with differing training datasets.

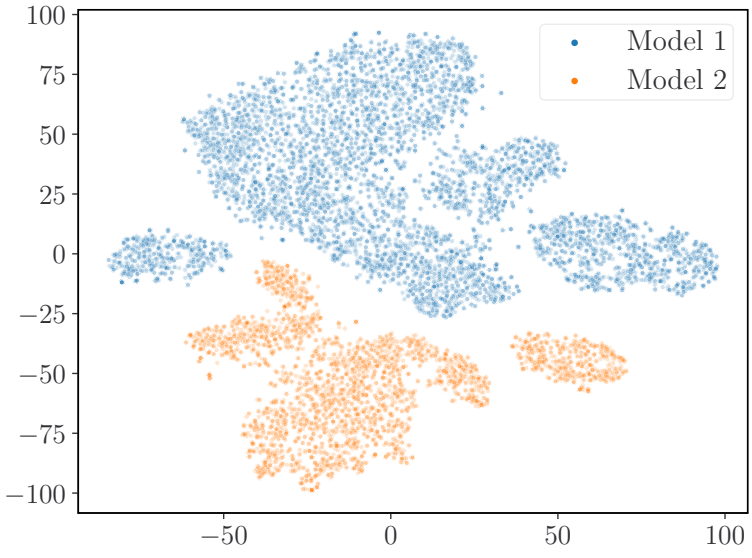
We pass $\mathcal{D}_v$ to both models to generate embeddings and visualize their t-SNE projections. The graphs for CoAuthor are shown in Figure 4.1, and the rest of the graphs are in Appendix A.1. We found that in every case, the t-SNE projections of the embeddings generated from $\mathcal{D}_v$ are distinguishable, even when the two models use the same dataset and architecture. This shows us that **embeddings cannot be used as a dataset fingerprint.**

Model Ownership. To check for model ownership, we use three models: $\mathcal{F}_t$, $\mathcal{F}_s$, $\mathcal{F}_i$, which may or may not share the same architecture and training data as $\mathcal{F}_t$. We set up a similar experiment to the one before. We split the dataset into two training datasets: $\mathcal{D}_1$ and $\mathcal{D}_2$. $\mathcal{D}_1$ is used to train both $\mathcal{F}_t$ and $\mathcal{F}_i$ since this is the worst-case scenario for triggering false accusations using the fingerprinting scheme. $\mathcal{F}_s$ is derived from $\mathcal{F}_t$ using $\mathcal{D}_2$ with Shen et al.'s [55] state-of-the-art GNN model extraction attack described in Section 2.2. Similar to the previous experiment, we build multiple combinations of the three models. We use three architectures for $\mathcal{F}_t$ and $\mathcal{F}_i$, and two for $\mathcal{F}_s$, resulting in 108 model combinations across six datasets.

We plot the embeddings of $\mathcal{D}_v$ from each of the three models and show the graphs for each dataset in Figure 4.2. We found that regardless of the architecture or the training data, the t-SNE projections of $\mathcal{H}_t$ and $\mathcal{H}_s$ are similar but distinct from $\mathcal{H}_i$. This motivates our choice to use embeddings as a fingerprint for model ownership.



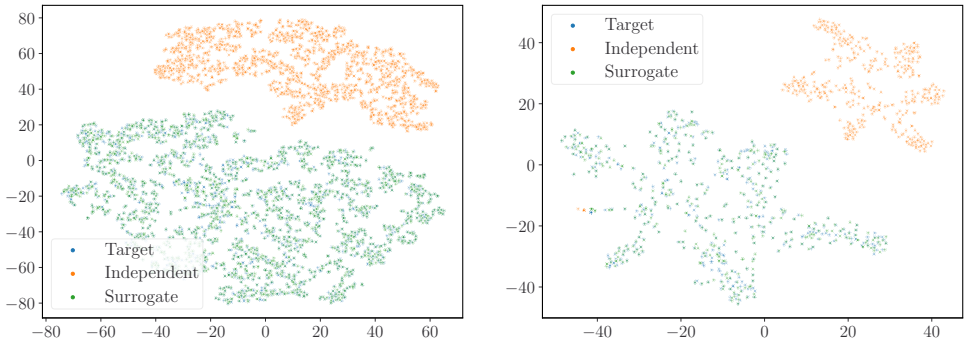
(a) Different Training Data



(b) Same Training Data

Figure 4.1: t-SNE projections of the embeddings from two models trained on CoAuthor using both the same and different architecture. The embeddings are distinguishable even when the architecture is the same.. The graphs for the other datasets can be found in Appendix A

(a) Models trained on CoAuthor dataset (b) Models trained on DBLP dataset



(c) Models trained on Pubmed dataset (d) Models trained on Citeseer dataset

(e) Models trained on ACM dataset (f) Models trained on Amazon dataset

Figure 4.2: t-SNE projections of the embeddings from $\mathcal{F}_s$ and $\mathcal{F}_t$ overlap, while those from $\mathcal{F}_i$ are distinct. The models in this plot are all trained with the GAT architecture.

Chapter 5

GROVE: Graph Embeddings for Model Ownership Verification

We now describe the design of GROVE, which uses embeddings as fingerprints.

Recall that Third-Party Verifier ($\mathcal{V}er$) aims to identify whether $\mathcal{F}_?$ was obtained via a model extraction attack on $\mathcal{F}_t$ (Section 3.1). As shown in Chapter 4, we know that the embeddings generated by $\mathcal{F}_t$ and $\mathcal{F}_s$ are similar. Our verification scheme relies on this observation and identifies whether the distances between $\mathcal{H}_?$ (generated by $\mathcal{F}_?$) and $\mathcal{H}_t$ are *close enough* to suggest a model extraction attack.

The simplest way to identify this is to calculate a distance metric between $\mathcal{H}_?$ and $\mathcal{H}_t$. If the aggregated distances are smaller than a tuned threshold, we can classify $\mathcal{F}_?$ and $\mathcal{F}_s$, and $\mathcal{F}_i$ otherwise. However, our experiments found that the distances between ($\mathcal{F}_t$, $\mathcal{F}_i$) pairs and ($\mathcal{F}_t$, $\mathcal{F}_s$) pairs overlapped, leading to high error rates. This phenomenon is visualized in the distance plots in Appendix B. Another simple approach is to compare the distributions of $\mathcal{H}_t$ and $\mathcal{H}_?$ using a Kolmogorov–Smirnov test (KS test) [41]. The KS test calculates the likelihood that two samples are drawn from the same probability distribution. However, we found that hypothesis testing also leads to high false positive rates for some datasets, and high false negative rates for other datasets.

Thus, similar to the intuition used for Siamese Networks [31], we use an Machine Learning (ML) classifier to classify a pair of embeddings as *similar* or *not similar*. An ML-based approach compresses the distance between similar data points and stretches the distance between dissimilar data points. The ML classifier is a multi-layer perceptron, denoted as $\mathcal{C}_{sim}$. Our verification scheme is divided into two phases: Phase 1 involves

training $\mathcal{C}_{sim}$; Phase 2 includes querying $\mathcal{C}_{sim}$ with pairs of embeddings generated by $\mathcal{F}_?$ and $\mathcal{F}_t$ on $\mathcal{D}_v$ to classify $\mathcal{F}_?$ as either a surrogate or independent model.

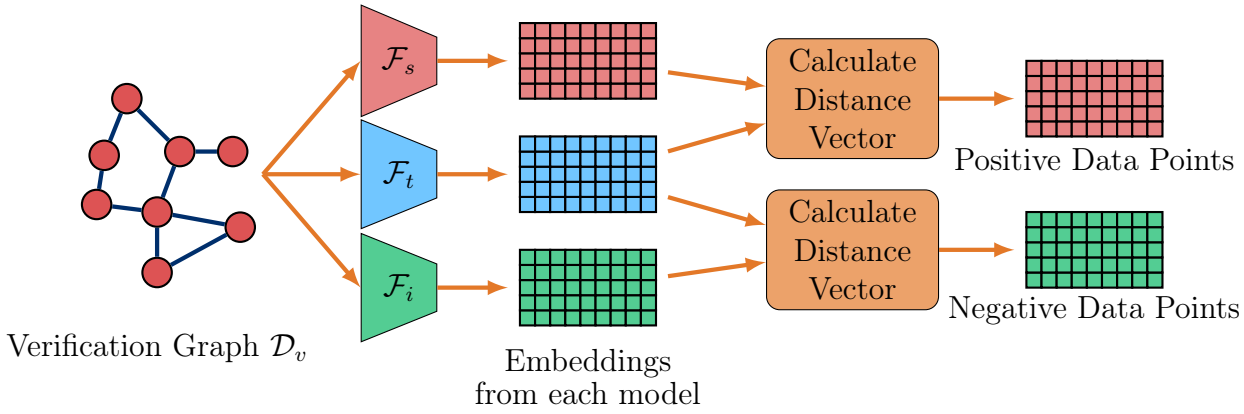


Figure 5.1: To generate training data for $\mathcal{C}_{sim}$, $\mathcal{D}_v$ is passed to $\mathcal{F}_t$, $\mathcal{F}_s$ and $\mathcal{F}_i$. We then compute the distance vectors between the embeddings obtained from each of the models, namely, ($\mathcal{F}_t$, $\mathcal{F}_s$) and ($\mathcal{F}_t$, $\mathcal{F}_i$) which act as positive and negative data points respectively.

$\mathcal{C}_{sim}$ Training. Figure 5.1 shows the process of generating training data for $\mathcal{C}_{sim}$. For each $\mathcal{F}_t$, we train multiple sets of $\mathcal{F}_s$ and $\mathcal{F}_i$ using different architectures. $\mathcal{V}er$ queries $\mathcal{F}_t$, $\mathcal{F}_s$ and $\mathcal{F}_i$ with $\mathcal{D}_v$ to generate embeddings. Each node in $\mathcal{D}_v$ will thus have three corresponding embeddings from each Graph Neural Network (GNN), denoted as h_t, h_s, and h_i, respectively. $\mathcal{V}er$ generates a distance vector between h_t and h_s, representing a positive (similar) data point. In contrast, the distance vector between h_t and h_i is represented as a negative (not similar) data point. The distance vector is the element-wise squared distance between the two embeddings. This allows $\mathcal{V}er$ to generate many data points from one pair of models equal to the size of $\mathcal{D}_v$. $\mathcal{V}er$ then uses this data to optimize $\mathcal{C}_{sim}$ to classify whether a pair of embeddings is similar.

Verification. Once $\mathcal{C}_{sim}$ has been trained $\mathcal{V}er$ can use it to make inferences about $\mathcal{F}_?$. $\mathcal{V}er$ queries both $\mathcal{F}_?$ and $\mathcal{F}_t$ with $\mathcal{D}_v$ to generate embeddings. The distance vector is calculated between corresponding embeddings and passed to $\mathcal{C}_{sim}$, which outputs whether the pair is similar. If more than 50% of the pairs of embeddings are classified as similar, $\mathcal{V}er$ classifies $\mathcal{F}_?$ as a surrogate, and independent otherwise. In our experiments, we found that the precise threshold does not matter. In general, for ($\mathcal{F}_t$, $\mathcal{F}_s$) pairs, $\approx 90\%$ of the embeddings are similar, while for ($\mathcal{F}_t$, $\mathcal{F}_i$) pairs, only $\approx 10\%$ embeddings are classified as similar. That being said, the threshold can be tuned using additional training data if needed.

Chapter 6

Experimental Setup

We evaluate GROVE using six datasets, three model architectures, and two model extraction attacks. We describe the datasets and their splits (Section 6.1), model architectures (Section 6.2), metrics for evaluation (Section 6.3), and the model extraction attacks (Section 6.4).

6.1 Datasets

Following prior work [55], we consider six benchmark graph datasets representing different types of graph networks. We describe the details of each of these datasets below.

DBLP [48] is a citation network where the 17,716 nodes represent publications and 105,734 edges indicate citations between different publications. Each node has 1,639 features based on the keywords in the paper. This is a node classification problem with four classes indicating the publication category.

CITESEER [17] is a citation network where the 4,120 nodes represent publications and 5,358 edges indicate citations between different publications. Each node has 602 features indicating the absence/presence of the corresponding word from the dictionary. This is a node classification task where the publications are categorized into six classes.

PUBMED [53] is a citation network where the 19,717 nodes represent publications and 88,648 edges indicate citations between different publications. Each node has 500 features described by a TF/IDF weighted word vector from a dictionary which consists of 500 unique words. This is a node classification task where the publications are categorized into three classes.

CoAuthor [54] is a co-authorship network where the 34,493 nodes represent different authors, which are connected with 495,924 edges if they coauthored a paper. Here, the 8,415 node features represent paper keywords for each author's papers. This node classification task is to predict the most active field of study out of five possibilities for each author.

ACM [73] is a heterogeneous graph that contains 3025 papers published in KDD, SIGMOD, SIGCOMM, MobiCOMM, and VLDB. Papers that the same author publishes have an edge between them, resulting in 26,256 edges. Each paper is divided into three classes (Database, Wireless Communication, Data Mining), and the 1,870 features for each node are the bag-of-words representation of their keywords.

Amazon [42] is an abbreviation for Amazon Co-purchase Network for Photos. The 7,650 nodes represent items, and the 143,663 edges indicate whether the two items are bought together. Each of the 745 nodes features are bag-of-words encoded product reviews. Each of the items is classified into one of eight product categories.

Dataset Splits for Model Extraction. We split the dataset into two disjoint sets, each 40% of the dataset. One chunk is used to train $\mathcal{F}_t$, and the other chunk is used to train $\mathcal{F}_s$. While the original model extraction attacks use overlapping and non-overlapping training data for $\mathcal{F}_s$, we choose a non-overlapping dataset since that is the most challenging case for Third-Party Verifier ($\mathcal{V}er$), as $\mathcal{F}_s$ is distinct from $\mathcal{F}_t$. For the same reason, $\mathcal{F}_t$ and $\mathcal{F}_i$ are trained on the same set. We evaluate $\mathcal{F}_t$ and $\mathcal{F}_s$ on a test set that is 10% of the dataset. Finally, $\mathcal{D}_v$ is the remaining 10%. The dataset sizes are shown in Table 6.1

Dataset	$\mathcal{F}_t$ Train	$\mathcal{F}_s$ Train	Test	$\mathcal{D}_v$
CoAuthor	13797	13797	3449	3449
Pubmed	7887	7887	1972	1972
DBLP	7086	7086	1772	1772
Amazon	3060	3060	765	765
Citeseer	1648	1648	412	412
ACM	1210	1210	303	303

Table 6.1: Data splits for training and evaluating different models.

6.2 Model Architectures

Following prior work [55], we use the Graph Attention Network (GAT), Graph Isomorphism Network (GIN), and Graph Sample and Aggregate (GraphSAGE) architectures

(Section 2.1). For all architectures, the hidden layer size is 256, and the final hidden layer is connected to a dense layer for classification. The embeddings are extracted from the last hidden layer. We use the cross-entropy loss as the node classification loss, the ReLU activation function, and the Adam optimizer with an initial learning rate of 0.001. All models are trained for 200 epochs with early stopping based on validation accuracy.

GIN. We use a three-layer GIN model. The neighborhood sample size is fixed at ten samples at each layer.

GAT. We use a three-layer GAT model with a fixed neighborhood sample size of 10 at each layer. The first and second layers have four attention heads each.

GraphSAGE. We use a two-layer GraphSAGE model. The neighborhood sample sizes are set to 25 and 10, respectively. Following prior work [19], the MEAN aggregation function is used at each layer, and the dropout is set to 0.5 to prevent overfitting.

Similarity Model. We use the Scikit-Learn implementation of an MLPClassifier to train $\mathcal{C}_{sim}$. Using the three architectures, we train three versions of $\mathcal{F}_t$ for each dataset. Each $\mathcal{F}_t$ has its own $\mathcal{C}_{sim}$. We train three versions of $\mathcal{F}_i$ and two versions of $\mathcal{F}_s$ for each $\mathcal{F}_t$ using different architectures. We use these models to generate the positive and negative data points as explained in Chapter 5. We use a two-layer MLP and find the best hyperparameters using a grid search. The hidden layer sizes in the search are either 64 or 128, and the activation function is either Tanh or ReLU. We select the best model based on 10-fold cross-validation. Unless otherwise stated, we only train $\mathcal{C}_{sim}$ on Type 1 attacks.

To evaluate $\mathcal{C}_{sim}$, we train nine different versions of $\mathcal{F}_i$ and $\mathcal{F}_s$, i.e., 45 test models with different architectures and random initialization. These additional models are used to ensure the statistical significance of the results. We ran each experiment five times and reported the average with a 95% confidence interval.

6.3 Metrics

Following prior work [55, 24], we use two metrics for evaluating the effectiveness of model extraction attacks: accuracy and fidelity.

Accuracy measures the number of predictions $\mathcal{F}_s$ classifies correctly, compared to the ground-truth.

Fidelity measures the agreement between $\mathcal{F}_t$ and $\mathcal{F}_s$.

To evaluate the effectiveness of GROVE, we use two additional metrics:

False Positive Rate (FPR) is the ratio of false positives to the sum of false positives and true negatives. This indicates the fraction of independent models incorrectly classified as surrogate.

False Negative Rate (FNR) is the ratio of false negatives to the sum of false negatives and true positives. This indicates the fraction of surrogate models incorrectly classified as independent.

6.4 Model Extraction Attack

We use Shen et al.'s [55] model extraction attack against inductive Graph Neural Networks (GNNs) from their source code[1]. The details of the attack have been explained in Section 2.2. As mentioned in Section 6.1, we train $\mathcal{F}_s$ on $\mathcal{D}_s$ that is disjoint from $\mathcal{D}_t$. All the hyperparameters are set to the default values presented in the original paper. The performance of $\mathcal{F}_t$ and $\mathcal{F}_i$ is summarized in Table 6.2 and the performance of the surrogate models is summarized in Table 6.3. Our results are similar to those reported in the original attack paper.

Table 6.2: Average accuracy (with 95% confidence intervals) of $\mathcal{F}_t$ and $\mathcal{F}_i$ used in the evaluation. Values are averaged across multiple architectures (Section 6.2).

Dataset	$\mathcal{F}_t$ Accuracy	$\mathcal{F}_i$ Accuracy
ACM	0.906 ± 0.025	0.919 ± 0.021
Amazon	0.879 ± 0.064	0.876 ± 0.050
Citeseer	0.804 ± 0.047	0.809 ± 0.028
CoAuthor	0.926 ± 0.005	0.928 ± 0.011
DBLP	0.696 ± 0.028	0.693 ± 0.030
Pubmed	0.846 ± 0.022	0.846 ± 0.021

[1] https://github.com/xinleihe/GNNStealing

Table 6.3: Average accuracy and fidelity (with 95% confidence intervals) of $\mathcal{F}_s$ derived from Type 1 and Type 2 attacks. Values are averaged across multiple architectures (Section 6.2).

Attack Type	**Dataset**	$\mathcal{F}_s$ **Accuracy**	$\mathcal{F}_s$ **Fidelity**
Type 1 Attack	ACM	0.888 ± 0.019	0.931 ± 0.019
	Amazon	0.861 ± 0.022	0.870 ± 0.051
	Citeseer	0.757 ± 0.014	0.907 ± 0.041
	CoAuthor	0.919 ± 0.019	0.949 ± 0.034
	DBLP	0.674 ± 0.009	0.833 ± 0.018
	Pubmed	0.829 ± 0.007	0.923 ± 0.016
Type 2 Attack	ACM	0.896 ± 0.010	0.954 ± 0.020
	Amazon	0.842 ± 0.007	0.848 ± 0.009
	Citeseer	0.796 ± 0.000	0.902 ± 0.012
	CoAuthor	0.919 ± 0.004	0.948 ± 0.003
	DBLP	0.680 ± 0.008	0.851 ± 0.017
	Pubmed	0.832 ± 0.005	0.937 ± 0.014

Chapter 7

Evaluation of GROVE

We show that GROVE satisfies the requirements outlined in Section 3.3. To this end, we evaluate GROVE's effectiveness (requirement **R2**) in differentiating between $\mathcal{F}_s$, derived from the two types of model extraction attacks, and $\mathcal{F}_i$ (Section 7.1). We evaluate GROVE's robustness (requirement **R3**) to $\mathcal{A}dv.\mathcal{R}$'s attempts at evading verification (Section 7.2). Finally, we evaluate GROVE's efficiency (requirement **R4**) (Section 7.3).

7.1 Effectiveness

Dataset	FPR	Type 1 FNR	Type 2 FNR
ACM	0.022 ± 0.022	0.000 ± 0.000	0.000 ± 0.000
AMAZON	0.034 ± 0.029	0.000 ± 0.000	0.000 ± 0.000
CITESEER	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
COAUTHOR	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
DBLP	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
PUBMED	0.002 ± 0.002	0.000 ± 0.000	0.000 ± 0.000

Table 7.1: Performance of GROVE against Type 1 and Type 2 attacks. Average FPR and FNR values are reported across 5 experiments with 95% confidence intervals.

We first show that GROVE is effective at distinguishing between $\mathcal{F}_i$ and $\mathcal{F}_s$ (requirement **R2**). We train three $\mathcal{F}_i$ models and two $\mathcal{F}_s$ models using the Type 1 attack to train $\mathcal{C}_{sim}$. During the testing phase of $\mathcal{C}_{sim}$, we train additional $\mathcal{F}_i$ models to compute the FPR and $\mathcal{F}_s$ models from both Type 1 and Type 2 attacks to compute the FNR.

Ideally, we expect GROVE to have low FPR and FNR while differentiating between $\mathcal{F}_i$ and $\mathcal{F}_s$ using $\mathcal{C}_{sim}$. As seen in Table 7.1, we find that GROVE indeed has zero false negatives against both model extraction attacks. We observe close to zero false positives across four datasets (DBLP, PUBMED, CITESEER, and COAUTHOR). For the AMAZON and ACM datasets, we note a low false positive rate of 3.4% and 2.2%, respectively.

These results show that GROVE is effective at distinguishing between $\mathcal{F}_s$ and $\mathcal{F}_i$ across different datasets and different architectures, satisfying **R2**.

7.2 Adversarial Robustness of GROVE

We now discuss the robustness of GROVE against attempts to evade detection by $\mathcal{A}dv.\mathcal{R}$ (requirement **R3**). $\mathcal{A}dv.\mathcal{R}$ can evade detection by differentiating $\mathcal{F}_s$ from $\mathcal{F}_t$ via (1) simple evasion or (2) model retraining.

Simple evasion techniques can be used by $\mathcal{A}dv.\mathcal{R}$ to evade detection without retraining $\mathcal{F}_s$: (a) *model replacement* and (b) *post-processing*. These are described below.

Model replacement occurs when during the verification process, $\mathcal{V}er$ intends to query the model that $\mathcal{R}esponder$ has deployed ($\mathcal{F}_s$), but $\mathcal{A}dv.\mathcal{R}$ replaces it by an independent model $\mathcal{F}_i$ to deceive $\mathcal{V}er$. Our system model (Section 3.1) pre-empts this by requiring $\mathcal{R}esponder$ to register $\mathcal{F}_s$ before deployment. During verification, $\mathcal{V}er$ conducts a fidelity check between the outputs of the registered and deployed models (i.e., both outputs should match perfectly) to confirm that they are the same. As both models should be identical, the fidelity score between embeddings of the same input should be perfect.

Post-processing occurs when $\mathcal{A}dv.\mathcal{R}$ applies a (linear) transformation on $\mathcal{H}_s$ to change its distribution while maintaining utility. If the distances between the embeddings of any two nodes of an input graph remain relatively the same after the transformation, it will not affect the result of any downstream task. Our system model pre-empts this by requiring $\mathcal{A}dv.\mathcal{R}$ to send $\mathcal{F}_s$ to $\mathcal{V}er$, who verifies whether the outputs are generated directly from $\mathcal{F}_s$ without being post-processed. Furthermore, $\mathcal{V}er$ can inspect $\mathcal{F}_s$ to ensure there are no non-standard layers that arbitrarily transform the output (Step 3 of the Verification Process in Section 3.1).

Model retraining. We identify three possible techniques from prior work to evade detection via model retraining [39, 50]: (a) *fine-tuning*, (b) *double extraction*, and (c) *pruning*.

Fine-tuning retrains a previously trained model on a new dataset to improve model performance or change the classification task by replacing the model's final layer. While

this is popular in prior work [39, 50, 40], we argue that it cannot be used by $\mathcal{A}dv.\mathcal{R}$ to evade detection of extracted GNN models. Recall from Section 6.4 that $\mathcal{F}_s$ consists of two independent components: the first outputs embeddings, and the second outputs class labels. Recalling (2.6), the embeddings are updated using an Mean Squared Error (MSE) loss with embeddings from $\mathcal{F}_t$. Traditional fine-tuning based on different class labels will only update the classifier, not affecting the embeddings.

Thus, we only test GROVE against end-to-end fine-tuning, where $\mathcal{A}dv.\mathcal{R}$ updates both the described components while fine-tuning on a separate dataset from the same distribution. The average performance of the models is reported in Table 7.2. We found that $\mathcal{F}_s$ accuracy improves slightly after end-to-end fine-tuning while the fidelity slightly decreases. Despite this change, GROVE still achieves zero false negatives across all datasets, showing GROVE is effective in mitigating end-to-end fine-tuning attacks across different datasets.

Attack Type	Dataset	$\mathcal{F}_s$ Accuracy	$\mathcal{F}_s$ Fidelity
Type 1 Attack	ACM	0.899 ± 0.030	0.927 ± 0.041
	AMAZON	0.875 ± 0.018	0.874 ± 0.029
	CITESEER	0.787 ± 0.018	0.838 ± 0.018
	COAUTHOR	0.935 ± 0.005	0.939 ± 0.005
	DBLP	0.706 ± 0.010	0.706 ± 0.021
	PUBMED	0.832 ± 0.010	0.924 ± 0.007
Type 2 Attack	ACM	0.902 ± 0.013	0.939 ± 0.021
	AMAZON	0.866 ± 0.021	0.863 ± 0.032
	CITESEER	0.765 ± 0.017	0.834 ± 0.031
	COAUTHOR	0.937 ± 0.006	0.940 ± 0.003
	DBLP	0.708 ± 0.011	0.724 ± 0.033
	PUBMED	0.841 ± 0.007	0.935 ± 0.011

Table 7.2: Performance of end-to-end fine-tuning. In many cases $\mathcal{F}_s$ accuracy is higher, but the fidelity is lower. For COAUTHOR the accuracy of $\mathcal{F}_s$ surpassed $\mathcal{F}_t$ after fine-tuning.

Double extraction involves $\mathcal{A}dv.\mathcal{R}$ running two model extraction attacks to obtain the final $\mathcal{F}_s$: first against $\mathcal{F}_t$ to get an intermediate model, followed by another attack against the intermediate model to obtain $\mathcal{F}_s$. This additional attack is to make the $\mathcal{F}_s$ distinct from $\mathcal{F}_t$. We refer to such surrogates as $\mathcal{F}_{s^2}$

To satisfy the robustness experiment, GROVE should achieve low FNR against double extraction even if $\mathcal{F}_{s^2}$ accuracy drops by up to 5% points. An accuracy drop greater than 5% points greatly reduces model utility. We use the previously trained $\mathcal{C}_{sim}$ directly and build additional $\mathcal{F}_{s^2}$ models on the previously trained $\mathcal{F}_t$ models. We use the same

attack for both extractions since the difference between the two attacks is the knowledge of $\mathcal{A}dv.\mathcal{R}$, and it is unrealistic for $\mathcal{A}dv.\mathcal{R}$ to have differing knowledge when running two attacks sequentially. As before, we use two architectures for each $\mathcal{F}_{s^2}$ per $\mathcal{F}_t$, and create nine versions using random initialization (a total of 18 test models per $\mathcal{F}_t$). We repeat each experiment five times.

We evaluated GROVE against $\mathcal{F}_{s^2}$ models reported in Table 7.3. We find that while the adversary experiences a significant loss in utility (at least 3% points in accuracy) on ACM, AMAZON, and CITESEER datasets, GROVE still achieves zero false negatives across all datasets. This shows that GROVE is effective in mitigating double extraction attacks against both Type 1 and Type 2 attacks and across different datasets.

Attack Type	Dataset	$\mathcal{F}_s$ Accuracy	$\mathcal{F}_s$ Fidelity
Type 1 Attack	ACM	0.843 ± 0.059	0.882 ± 0.060
	AMAZON	0.776 ± 0.050	0.781 ± 0.063
	CITESEER	0.551 ± 0.140	0.627 ± 0.159
	COAUTHOR	0.924 ± 0.005	0.947 ± 0.012
	DBLP	0.686 ± 0.011	0.783 ± 0.011
	PUBMED	0.830 ± 0.007	0.912 ± 0.007
Type 2 Attack	ACM	0.882 ± 0.017	0.930 ± 0.020
	AMAZON	0.698 ± 0.216	0.695 ± 0.219
	CITESEER	0.679 ± 0.064	0.736 ± 0.093
	COAUTHOR	0.916 ± 0.009	0.943 ± 0.004
	DBLP	0.678 ± 0.019	0.784 ± 0.036
	PUBMED	0.831 ± 0.004	0.930 ± 0.005

Table 7.3: Performance of $\mathcal{F}_{s^2}$. The model utility is comparable to $\mathcal{F}_s$ in most cases, but drops by $\approx 8\%$ points for AMAZON and $\approx 12\%$ points for CITESEER.

Pruning removes model weights to reduce computational complexity while maintaining model utility. This alters the output distribution (in our case, embeddings), which could potentially affect the success of GROVE.

We experiment with *prune ratios* (ratio of weights set to 0) ranging from 0.1 to 0.7 as pruning beyond resulted in a high accuracy loss (>20% points for all datasets). As before, GROVE is robust if it achieves low FNR against $\mathcal{F}_s$ models with less than a 5% point drop in accuracy. We use the same experimental setup of training nine versions of $\mathcal{F}_s$ and pruning each one with the ratios above (a total of 18 test models per prune ratio).

We report our results in Figure 7.1. We observe that $\mathcal{F}_s$ accuracy falls significantly after 0.4 (>5% points for all datasets). Only the FNR for COAUTHOR stays small until a

ratio of 0.6. We conjecture this is due to CoAuthor being the largest dataset, it has 1.8 times more nodes and 5 times more edges than the second largest dataset. For the rest of the datasets the FNR increases around a ratio of 0.2 to 0.3. This shows that GrOVe in its basic form fails against pruning.

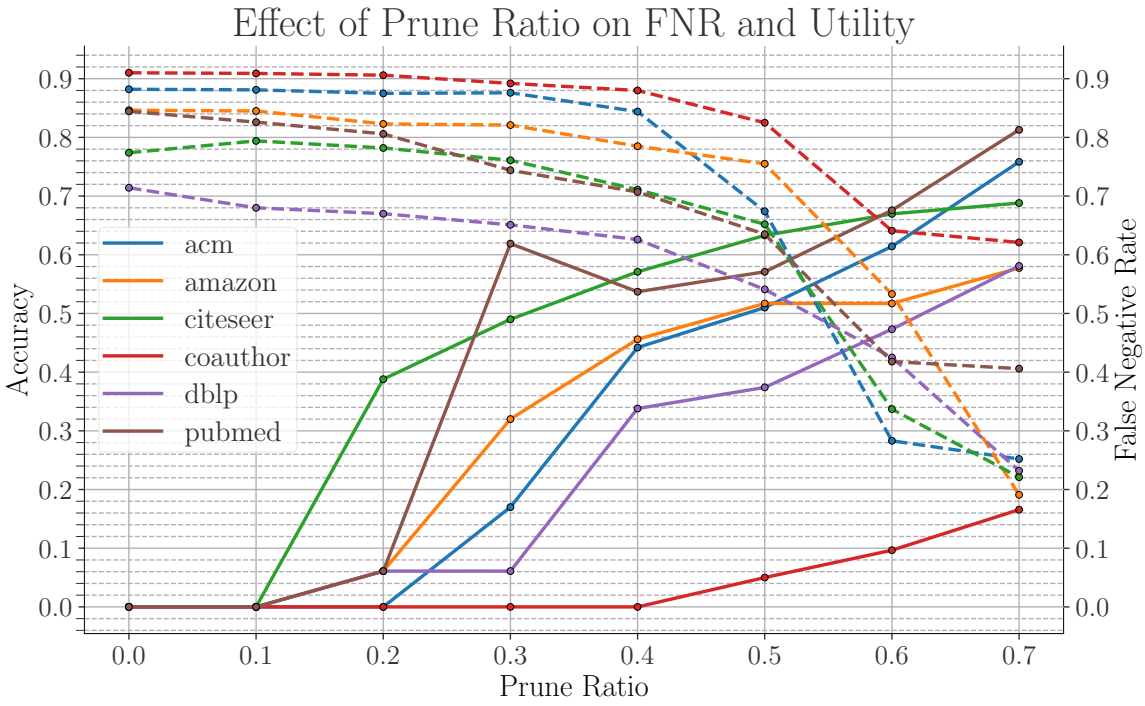


Figure 7.1: GrOVe performance against pruning. Dotted lines represent $\mathcal{F}_s$ accuracy, and solid lines represent FNR. As the pruning ratio increases, the accuracy decreases, and the FNR increases. By default, GrOVe fails against prune ratios of 0.3-0.4 for most datasets.

GrOVe can be made robust against pruning attacks by using data augmentation techniques while training $\mathcal{C}_{sim}$ (Chapter 5). We do this by including the output from pruned models up to a ratio of 0.4 into the training data. Beyond a prune ratio of 0.4, the accuracy drop is large enough ($> 5\%$ points) to deter $\mathcal{A}dv.\mathcal{R}$. Using the same experimental setup, we evaluate the success of the more robust GrOVe to mitigate pruning.

We observe that GrOVe still achieves zero false negatives against the basic and double extraction attacks mentioned before. Additionally, it achieves a lower FPR, with only a 1.4% FPR for Amazon, 0.7% FPR for Pubmed, and a 0% FPR for the other datasets. We report the results for pruning in Figure 7.2. It achieves nearly zero FNR for all datasets

up to a prune ratio of 0.4. Noting the success of robust optimization, we conjecture that this can be used to make GROVE robust against future evasion techniques.

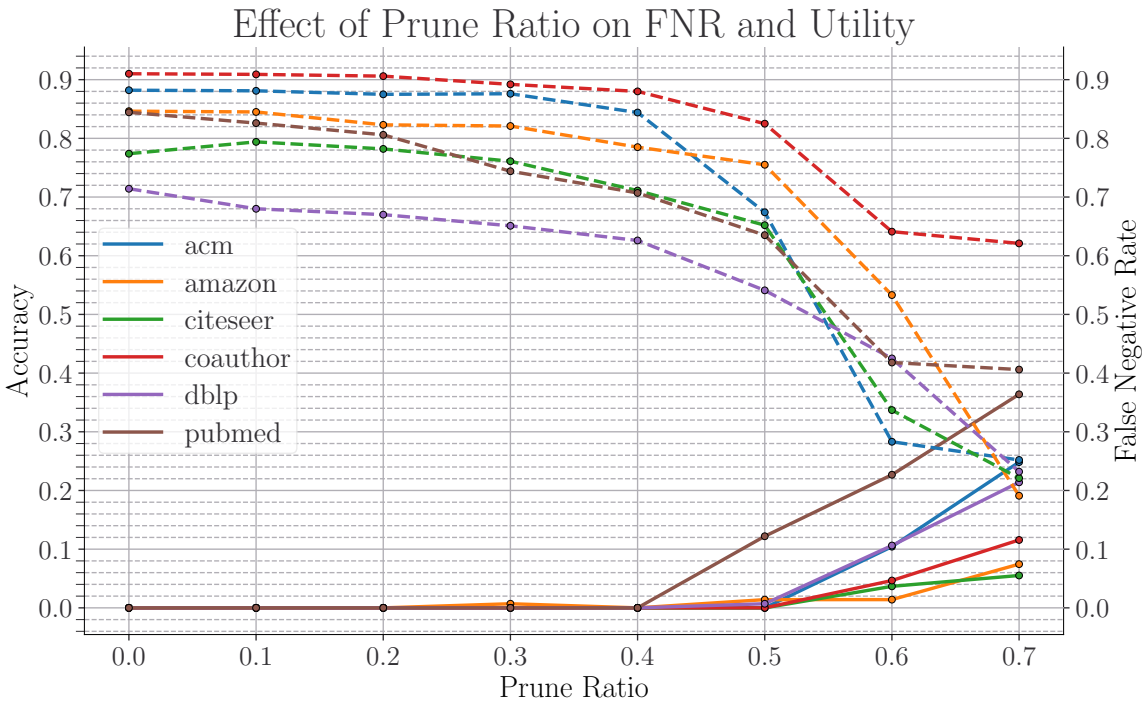


Figure 7.2: A more robust GROVE performs better against pruning. Dotted lines represent $\mathcal{F}_s$ accuracy, and solid lines represent FNR. GROVE detects $\mathcal{F}_s$ without false negatives up to a prune ratio of 0.4. Beyond that, $\mathcal{F}_s$ utility decreases more than 5% points, thereby removing the incentive for rational attackers to steal the model.

We, therefore, conclude that GROVE remains effective (**R2**) even in the presence of adversaries (**R3**). Furthermore, data augmentation allows $\mathcal{V}er$ to choose between resource utilization and performance. The larger and more diverse the training data for $\mathcal{C}_{sim}$, the longer it would take to train GROVE, and the more robust it would be. We have shown the minimum training data required to perform well against the evasion techniques we identified.

Dataset	Architectures		
	GAT	GIN	GraphSAGE
CoAuthor	10562 ± 1548	10668 ± 1205	10550 ± 1237
Pubmed	3961 ± 492	3845 ± 257	3856 ± 288
DBLP	4182 ± 462	4011 ± 498	3730 ± 202
Amazon	3312 ± 218	3273 ± 171	3473 ± 323
Citeseer	3312 ± 124	3142 ± 204	2970 ± 186
ACM	2985 ± 165	2943 ± 223	2876 ± 134

Table 7.4: Average total time over five runs in seconds (with 95% confidence intervals) taken to generate training data and train $\mathcal{C}_{sim}$.

7.3 Efficiency of GROVE

We now evaluate the efficiency of GROVE (requirement **R4**). We want to ensure the computation overhead of GROVE is reasonable.

To this end, we measure the execution time to generate the training data for $\mathcal{C}_{sim}$ and train it. Recall from Section 6.2 that we train multiple versions of $\mathcal{F}_i$ and $\mathcal{F}_s$ to generate the training data for $\mathcal{C}_{sim}$. For GROVE, this involves building six versions of $\mathcal{F}_i$ (two models per architecture with different random initializations) and two versions of $\mathcal{F}_s$ along with their pruned variants (10 versions of $\mathcal{F}_s$). These models can be trained in parallel, making it faster. However, we train them sequentially to accurately measure the total time taken, assuming just one available machine. The models were trained on a machine with two AMD EPYC 7302 16-Core CPUs and eight Nvidia A100 GPUs with 40GB VRAM per GPU.

We summarize the results for each target model in Table 7.4. It generally takes less than 3 hours to train GROVE. The time taken to train GROVE includes the time taken to train the additional surrogate and independent models to generate the training data and to train $\mathcal{C}_{sim}$. The size of the dataset influences the time taken to train the additional models. CoAuthor is the largest dataset, with the longest training times, followed by DBLP and Pubmed. The variation in time taken within datasets is caused by the optimization of $\mathcal{C}_{sim}$. For instance, for Amazon, the time taken to train $\mathcal{C}_{sim}$ for GAT-based $\mathcal{F}_t$ was 187 seconds on average, while for GIN-based $\mathcal{F}_t$ it took 340 seconds on average. There is no trend for which architecture takes the longest time; it is affected by the combination of dataset and architecture. GROVE built for ACM-based models was the fastest to train, with $\mathcal{C}_{sim}$ training taking less than 100 seconds. For CoAuthor, however, $\mathcal{C}_{sim}$ training took much longer, between 1561 to 2103 seconds.

Recall that in the practical setting we describe in Section 3.1, $\mathcal{V}er$ only trains GROVE when $\mathcal{A}ccuser$ initiates a dispute with $\mathcal{R}esponder$. Considering most models will not encounter ownership disputes, we consider the numbers in Table 7.4 to be reasonable.

In conclusion, we show that GROVE satisfies the requirement of being an effective (**R2**) robust (**R3**) and efficient (**R4**) fingerprinting scheme.

Chapter 8

Related Works

In addition to the prior work directly related to GrOVe (Section 2.2), we broadly discuss other related work pertaining to privacy and security in GNNs, and model extraction attacks in other domains.

8.1 Robustness Attacks against GNNs

These attacks find adversarial examples modifying the adjacency matrix to perturb the input, which results in misclassification [78, 9, 15, 96, 97]. Against image-based Machine Learning (ML) models, this is often divided into targeted and untargeted attacks [13]. The goal of the adversary is to perturb a target node such that it is misclassified to any class (untargeted attack) or to a target class (targeted attack). As mentioned in Section 3.4, finding adversarial examples for graphs is not trivial. Since graph data is relational, any perturbation applied to either a node or an edge affects multiple nodes [96]. This is both a challenge and an opportunity; an attacker can influence the prediction of a target node without directly perturbing that node [96], but staying stealthy is difficult since many nodes might be affected by a single change. In the current literature, directly perturbing the edges of the target node is considered the most viable approach [96, 9, 78, 15].

The primary challenge in perturbing edges is that the adjacency matrix is a binary matrix. Thus, an attacker can only make discrete changes by adding or removing an edge (changing a 1 to 0 or vice versa). Finding the most effective perturbations in a discrete search space is difficult compared to a continuous search space (such as in images) [78, 9, 96]. Furthermore, adversarial perturbations are constrained to be imperceptible. This is easily

achieved in images by ensuring the difference between the perturbed image and the original image is small. In the discrete graph domain, this is achieved by preserving the degree distribution of the graph [96, 97]. Despite this progress, the best attack rate in these prior works is only 35%. For comparison, image-based robustness attacks can reach nearly 100% attack success rate [13].

8.2 Defenses against Robustness Attacks

Most defenses limit the effect of perturbed edges on the GNN [86, 11, 15, 44, 78, 87, 93] or use adversarial training (training with adversarial examples) to make the model robust [25, 63].

To limit the effect of perturbed edges, some defenses prune edges in the adjacency matrix to minimize the effect of adversarial perturbations on edges. Since homophily ensures nodes and the edges connecting them are similar, any non-similar edges are potentially adversarial and can be pruned [78, 86]. A similar approach is to compute the low-rank approximation of the adjacency and features matrix and then train the GNN on this approximation [15]. This can also be extended to cases where homophily is not satisfied by learning a reduced-rank estimation of the input graph [11]. RobustGCN [93] trains the model such that the embeddings follow a Gaussian distribution to reduce the effect of perturbed edges.

Adversarial training for GNNs is also more challenging than the image domain. Since the perturbations are discrete, finding the most effective perturbations is NP-hard [25]. Thus, one approach is to perturb the embeddings rather than the nodes while training adversarially [25]. Another approach is to include an additional loss function while training the GNN that penalizes potentially perturbed edges and reduces their weight in the final prediction [63]. However, this approach has not been tested on the state-of-the-art robustness attack [97]. In general, research on robustness attacks and defenses for GNNs is still in its early phases and is limited in applicability.

8.3 Privacy Attacks against GNNs

Given some background information and access to the model, adversaries can infer sensitive unobservable information pertaining to the training dataset. Membership inference attacks infer whether a node/sub-graph was used to train the model [45, 21, 14, 76]. Olatunji

et al. [45] present a node-level membership inference attack using the 2-hop subgraph of the target node. He et al. [21] achieve a similar goal, except they use only the 0-hop subgraph. Wu et al. [76] present the first work on graph-level membership inference attack. They use a shadow model approach to train an attack model for graph-level membership inference. Zhang et al. introduce a subgraph inference attack using a similar shadow model technique [89]. Membership inference attacks are, however, not the only privacy concern for GNNs.

Adversaries can also mount property inference attacks to extract information about the training data of the GNN. General properties of the training graph, such as the number of nodes, number of edges, graph density, etc., can be extracted through a multi-task classification model trained using shadow models [89]. More specific property inference attacks include identifying edges between two query nodes [20, 14], or extracting sensitive attributes of a node (such as gender) [14]. Prior work has also shown that entire graphs can be reconstructed from the embeddings using an encoder-decoder architecture [14, 89]. Property inference attacks designed for traditional Deep Neural Networks (DNNs) may also be used on GNNs [58, 59]. Considering the widespread use of graphs for storing sensitive information, such as social networks or medical data, these attacks show that model deployers should be cognizant of the privacy leakage in GNNs.

Chapter 9

Discussion

We discuss some implications of our system model assumptions and design choices as well as potential limitations of GrOVe.

9.1 Low-Fidelity Model Extraction Attacks

Recall that Shen et al.'s [55] attack aims to maximize fidelity between $\mathcal{F}_t$ and $\mathcal{F}_s$. Consequently, GrOVe relies on the fact that the embeddings of $\mathcal{F}_t$ and $\mathcal{F}_s$ are similar. Low-fidelity extraction attacks will not necessarily result in $\mathcal{F}_s$ being close to $\mathcal{F}_t$, and can be potentially missed by GrOVe. However, there are no low-fidelity GNN extraction attacks in the current literature. Thus, finding a low-fidelity model extraction attack could be an important direction for future work.

9.2 Evasion using Distribution Shift

Recall from Section 7.2 that our system model preempts simple attempts to induce a distribution shift on the output embeddings via model replacement. However, $\mathcal{A}dv.\mathcal{R}$ can adversarially train $\mathcal{F}_s$ to change the output distribution to make the embeddings distinct from $\mathcal{F}_t$'s embeddings. This is constrained to minimize degradation of $\mathcal{F}_s$ accuracy.

We designed an experiment to reproduce this by training $\mathcal{F}_s$ with an adversarial autoencoder that shifts the output distribution of $\mathcal{F}_s$ to a Gaussian distribution. In this, $\mathcal{F}_s$ is treated as a generator and is trained simultaneously with a discriminator that detects

the Gaussian distribution. While this should work in theory, we could not successfully modify the output distribution of $\mathcal{F}_s$. Our experiment found that GROVE could successfully classify the surrogate models. Furthermore, we found that the output distribution did not change a lot, as shown in Figure 9.1. We conjecture that the high-fidelity requirement forces the output distribution of $\mathcal{F}_s$ to be similar to $\mathcal{F}_t$. Thus, we could not find a successful $\mathcal{F}_s$ in the trade-off between evasiveness and utility.

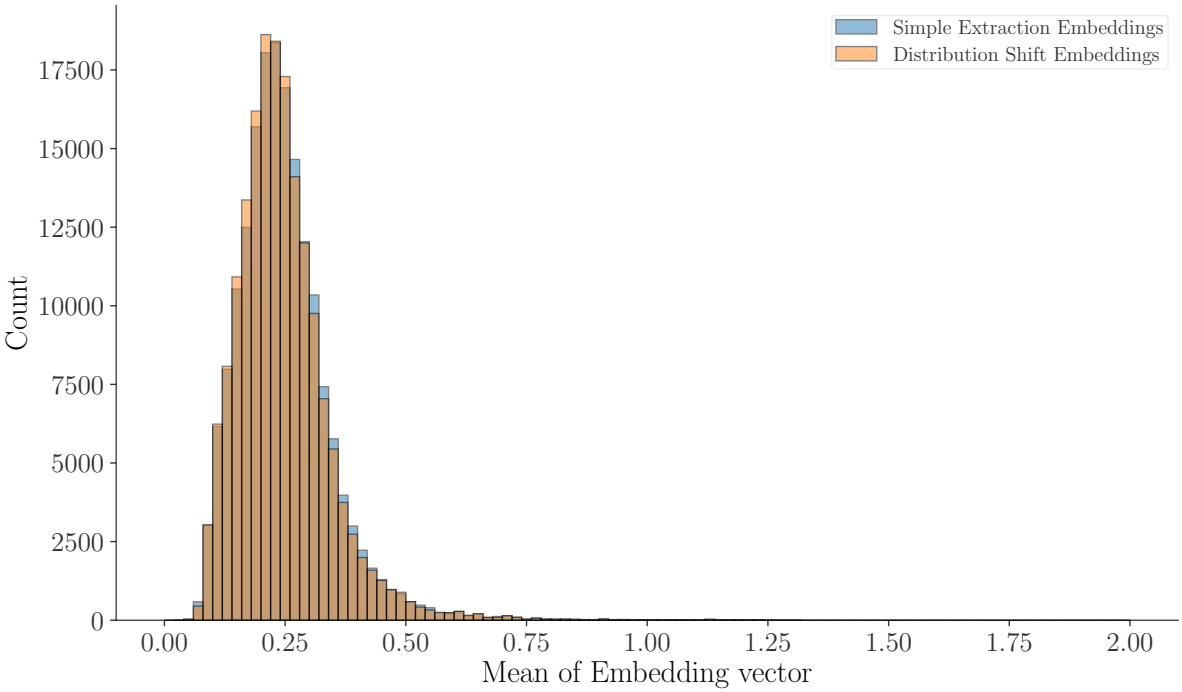


Figure 9.1: The mean of each embedding vector generated from surrogate models either trained via the basic extraction technique from [55], or through the distribution shift method. The embeddings are all on models trained on the PUBMED dataset using the GAT architecture. While the histograms of the mean values are slightly different, showing the distribution does change slightly, but not by much.

We further hypothesize that since we use an ML-based approach for GROVE, the best way of evading detection is to shift the distribution of the embeddings from $\mathcal{F}_s$, such that it matches the distribution of the embeddings from $\mathcal{F}_i$. However, an adversary would need to train their own independently trained models to achieve this. If an adversary can independently train a GNN that works just as well as $\mathcal{F}_t$, there is no incentive to use a

model extraction attack in the first place. We leave this exploration to future work.

9.3 Requirement for Model Registration

Following prior work [60, 1, 37], in our system model (Section 3.1), all model owners are required to obtain secure certified timestamps for their models before deployment to protect against different types of adversarial behavior. Model owners can be incentivized to do so because:

- registration serves as a direct means of protecting models against theft,
- it may be necessary to meet regulatory requirements for deployed models (e.g., EU guidelines[1]),
- in the future, a model insurance mechanisms may be used to mitigate the effects of model theft, and model registration may become a requirement for insurance.

9.4 Revisiting $\mathcal{A}dv.\mathcal{A}$

In Section 3.1, we argued that $\mathcal{A}dv.\mathcal{A}$ is thwarted by the model registration requirement. An alternative approach is to generate the fingerprint in such a way that it triggers false positives against all independent models [37]. In all prior model ownership schemes (both fingerprinting and watermarking) [39, 50], this is potentially feasible since it is the $\mathcal{A}ccuser$ who generates the fingerprint/watermark. Thus, if the $\mathcal{A}ccuser$ generates adversarial fingerprints, they can abuse the verification scheme to profit from false claims. This is not the case in GrOVe: the fingerprint is chosen by $\mathcal{V}er$, rendering GrOVe immune to any such adversarial fingerprint generation attacks.

9.5 Shen et al.'s [55] Prediction-based Attack

In our adversary model (Section 3.2), we consider that $\mathcal{F}_t$ outputs embeddings as done in Shen et al. [55]. This allows us to use GrOVe over the outputs of $\mathcal{F}_t$, making it a black-box scheme. However, Shen et al.'s [55] also suggest a prediction-based model extraction

[1] https://artificialintelligenceact.eu/

attack for cases where $\mathcal{F}_t$ only outputs classification labels. In such a setting, GrOVe is still applicable if $\mathcal{V}er$ is given white-box access to $\mathcal{F}_t$. We tested GrOVe in this setting by carrying out a prediction-based model extraction attack and extracting the penultimate layer embeddings from $\mathcal{F}_s$. We found that GrOVe still achieved a low FNR without any retraining; a max of 0.042 on the Amazon dataset, and 0.012 on the DBLP dataset.

9.6 Post-processing Embeddings

The system model described in Section 3.1 prevents the adversary from applying arbitrary transformations to the embeddings. Prior work has shown adding Gaussian noise is effective at mitigating property inference attacks that use embeddings [89]. Thus, it would be interesting to see whether similar techniques can be used to thwart GrOVe. Note that while these techniques might thwart GrOVe, they would do so at the risk of adversely affecting the utility of the model. The tradeoff between evasiveness and utility is yet to be explored.

Chapter 10

Conclusion

This thesis focuses on the challenges of ownership verification for GNNs.

Problem setup. We hypothesized that node embeddings generated by GNNs might be useful for ownership verification. We listed the desiderata for an effective ownership verification technique and identified the potential adversaries. Adversaries that wish to steal a model and evade detection are well-studied in prior work. We further identified adversaries that may abuse the verification process to maliciously accuse innocent model owners of stealing their models. Thus, we presented a system model that protects innocent model owners against both types of adversaries.

Potential solution. We identified why prior fingerprinting schemes that use adversarial examples are difficult to apply to GNNs. To motivate our hypothesis, we visualized the embeddings generated from the target model ($\mathcal{F}_t$), the surrogate model derived from a model stealing attack against $\mathcal{F}_t$ ($\mathcal{F}_s$), and an independently trained model ($\mathcal{F}_i$). The visualizations showed that embeddings from $\mathcal{F}_s$ and $\mathcal{F}_t$ form a cluster different from $\mathcal{F}_i$, even when $\mathcal{F}_i$ uses the same training data and architecture as $\mathcal{F}_t$. Using this insight, we designed an ML-based fingerprinting scheme, GrOVe, that utilizes the distance between pairs of embeddings from $\mathcal{F}_t$ and $\mathcal{F}_?$ to identify whether $\mathcal{F}_?$ is a surrogate of $\mathcal{F}_t$ or an independently trained model.

Evaluating the solution. We evaluated GrOVe's effectiveness against the state-of-the-art model extraction attacks for GNNs [55] and against possible evasion techniques. We found that GrOVe is effective, robust to attempts at evasion, and computationally efficient. GrOVe does not require retraining models to protect them, and the verification process only begins once a dispute is initiated. Furthermore, since $\mathcal{V}er$ generates the fingerprints, it is immune to malicious accusers that try to abuse the verification process.

Lastly, while we only tested data augmentation against pruning, we believe that data augmentation has the potential to make GROVE more robust as more model extraction attacks are introduced in the literature.

Future work. As pointed out in Section 9.2, it is theoretically possible for an adversary to evade GROVE by shifting the distribution of the embeddings generated by $\mathcal{F}_s$. Furthermore, there is still little work on model extraction attacks against GNNs. Thus, exploring the boundaries of GROVE is an important direction for future work. Some potential methods include low-fidelity model extraction attacks, a surrogate training mechanism that shifts the distribution of the output without requiring another independently trained GNN, or exploring how post-processing the embeddings impacts the verification process. Finally, implementing the privacy-preserving mechanisms to share models with $\mathcal{V}er$ is also left to future work.

References

[1] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1615–1631, Baltimore, MD, August 2018. USENIX Association.

[2] Buse Gul Atli, Sebastian Szyller, Mika Juuti, Samuel Marchal, and N. Asokan. Extraction of complex dnn models: Real threat or boogeyman? In Onn Shehory, Eitan Farchi, and Guy Barash, editors, *Engineering Dependable and Secure Machine Learning Systems*, pages 42–57, Cham, 2020. Springer International Publishing.

[3] Dan Boneh, Wilson Nguyen, and Alex Ozdemir. Efficient functional commitments: How to commit to a private function. Cryptology ePrint Archive, Paper 2021/1342, 2021.

[4] Lei Cai, Jundong Li, Jie Wang, and Shuiwang Ji. Line graph neural networks for link prediction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(9):5103–5113, 2021.

[5] Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Ipguard: Protecting intellectual property of deep neural networks via fingerprinting the classification boundary. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, ASIA CCS '21, page 14–25, New York, NY, USA, 2021. Association for Computing Machinery.

[6] Nicholas Carlini, Matthew Jagielski, and Ilya Mironov. Cryptanalytic extraction of neural network models. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020*, pages 189–218, Cham, 2020. Springer International Publishing.

[7] Ashutosh Chaubey, Nikhil Agrawal, Kavya Barnwal, Keerat K. Guliani, and Pramod Mehta. Universal adversarial perturbations: A survey. *CoRR*, abs/2005.08087, 2020.

[8] 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, page 105–113, New York, NY, USA, 2019. Association for Computing Machinery.

[9] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. Adversarial attack on graph structured data. *CoRR*, abs/1806.02371, 2018.

[10] David DeFazio and Arti Ramesh. Adversarial model extraction on graph neural networks. *arXiv preprint arXiv:1912.07721*, 2019.

[11] Chenhui Deng, Xiuyu Li, Zhuo Feng, and Zhiru Zhang. GARNET: Reduced-rank topology learning for robust and scalable graph neural networks. In *The First Learning on Graphs Conference*, 2022.

[12] Yunjie Deng, Chenxu Wang, Shunchang Yu, Shiqing Liu, Zhenyu Ning, Kevin Leach, Jin Li, Shoumeng Yan, Zhengyu He, Jiannong Cao, and Fengwei Zhang. Strongbox: A gpu tee on arm endpoints. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 769–783, New York, NY, USA, 2022. Association for Computing Machinery.

[13] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. Boosting adversarial attacks with momentum. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[14] Vasisht Duddu, Antoine Boutet, and Virat Shejwalkar. Quantifying privacy leakage in graph embedding. In *MobiQuitous 2020-17th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, pages 76–85, 2020.

[15] 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 *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 169–177, 2020.

[16] Edgar N Gilbert. Random graphs. *The Annals of Mathematical Statistics*, 30(4):1141–1144, 1959.

[17] C. L. Giles, K. D. Bollacker, and S. Lawrence. Citeseer: an automatic citation indexing system. In *Proceedings of the ACM International Conference on Digital Libraries*, pages 89–98. ACM, 1998. Proceedings of the 1998 3rd ACM Conference on Digital Libraries ; Conference date: 23-06-1998 Through 26-06-1998.

[18] Jia Guo and Miodrag Potkonjak. Watermarking deep neural networks for embedded systems. In *Proceedings of the International Conference on Computer-Aided Design*, ICCAD '18, New York, NY, USA, 2018. Association for Computing Machinery.

[19] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 1025–1035, Red Hook, NY, USA, 2017. Curran Associates Inc.

[20] Xinlei He, Jinyuan Jia, Michael Backes, Neil Zhenqiang Gong, and Yang Zhang. Stealing links from graph neural networks. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2669–2686. USENIX Association, August 2021.

[21] Xinlei He, Rui Wen, Yixin Wu, Michael Backes, Yun Shen, and Yang Zhang. Node-level membership inference attacks against graph neural networks, 2023.

[22] 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.

[23] Hailong Hu and Jun Pang. Stealing machine learning models: Attacks and countermeasures for generative adversarial networks. In *Annual Computer Security Applications Conference*, ACSAC '21, page 1–16, New York, NY, USA, 2021. Association for Computing Machinery.

[24] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. *High Accuracy and High Fidelity Extraction of Neural Networks*. USENIX Association, USA, 2020.

[25] Hongwei Jin and Xinhua Zhang. Latent adversarial training of graph convolution networks. In *ICML workshop on learning and reasoning with graph-structured representations*, volume 2, 2019.

[26] Mika Juuti, Sebastian Szyller, Alexey Dmitrenko, Samuel Marchal, and N. Asokan. PRADA: protecting against DNN model stealing attacks. *CoRR*, abs/1805.02628, 2018.

[27] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. Gazelle: A low latency framework for secure neural network inference. In *Proceedings of the 27th USENIX Conference on Security Symposium*, SEC'18, page 1651–1668, USA, 2018. USENIX Association.

[28] Daniel Kang, Tatsunori Hashimoto, Ion Stoica, and Yi Sun. Scaling up trustless DNN inference with zero-knowledge proofs. *CoRR*, abs/2210.08674, 2022.

[29] Manish Kesarwani, Bhaskar Mukhoty, Vijay Arya, and Sameep Mehta. Model extraction warning in mlaas paradigm. In *Proceedings of the 34th Annual Computer Security Applications Conference*, ACSAC '18, page 371–380, New York, NY, USA, 2018. Association for Computing Machinery.

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

[31] Gregory Koch, Richard Zemel, Ruslan Salakhutdinov, et al. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2. Lille, 2015.

[32] Kalpesh Krishna, Gaurav Singh Tomar, Ankur P. Parikh, Nicolas Papernot, and Mohit Iyyer. Thieves on sesame street! model extraction of bert-based apis. In *International Conference on Learning Representations*, 2020.

[33] 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, 2019.

[34] Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. PyTorch-BigGraph: A Large-scale Graph Embedding System. In *Proceedings of the 2nd SysML Conference*, Palo Alto, CA, USA, 2019.

[35] Chong Li, Kunyang Jia, Dan Shen, C.J. Richard Shi, and Hongxia Yang. Hierarchical representation learning for bipartite graphs. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 2873–2879. International Joint Conferences on Artificial Intelligence Organization, 7 2019.

[36] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 619–631, New York, NY, USA, 2017. Association for Computing Machinery.

[37] Jian Liu, Rui Zhang, Sebastian Szyller, Kui Ren, and N. Asokan. False claims against model ownership resolution, 2023.

[38] Nils Lukas, Edward Jiang, Xinda Li, and Florian Kerschbaum. Sok: How robust is image classification deep neural network watermarking? (extended version). *CoRR*, abs/2108.04974, 2021.

[39] Nils Lukas, Yuxuan Zhang, and Florian Kerschbaum. Deep neural network fingerprinting by conferrable adversarial examples. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021.

[40] Pratyush Maini, Mohammad Yaghini, and Nicolas Papernot. Dataset inference: Ownership resolution in machine learning. In *International Conference on Learning Representations*, 2021.

[41] Frank J. Massey. The kolmogorov-smirnov test for goodness of fit. *Journal of the American Statistical Association*, 46(253):68–78, 1951.

[42] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, page 43–52, New York, NY, USA, 2015. Association for Computing Machinery.

[43] Erwan Merrer, Patrick Perez, and Gilles Trédan. Adversarial frontier stitching for remote neural network watermarking. *Neural Computing and Applications*, 32, 07 2020.

[44] Benjamin A Miller, Mustafa Çamurcu, Alexander J Gomez, Kevin Chan, and Tina Eliassi-Rad. Improving robustness to attacks against vertex classification. In *MLG Workshop*, 2019.

[45] Iyiola E. Olatunji, Wolfgang Nejdl, and Megha Khosla. Membership inference attack on graph neural networks. In *2021 Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, pages 11–20, 2021.

[46] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. Knockoff nets: Stealing functionality of black-box models. *CoRR*, abs/1812.02766, 2018.

[47] Soham Pal, Yash Gupta, Aditya Shukla, Aditya Kanade, Shirish Shevade, and Vinod Ganapathy. A framework for the extraction of deep neural networks by leveraging public data, 2019.

[48] Shirui Pan, Jia Wu, Xingquan Zhu, Chengqi Zhang, and Yang Wang. Tri-party deep network representation. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, page 1895–1901. AAAI Press, 2016.

[49] 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, page 506–519, New York, NY, USA, 2017. Association for Computing Machinery.

[50] Zirui Peng, Shaofeng Li, Guoxing Chen, Cheng Zhang, Haojin Zhu, and Minhui Xue. Fingerprinting deep neural networks globally via universal adversarial perturbations. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, pages 13420–13429. IEEE, 2022.

[51] Emanuele Rossi, Fabrizio Frasca, Ben Chamberlain, Davide Eynard, Michael M. Bronstein, and Federico Monti. SIGN: scalable inception graph neural networks. *CoRR*, abs/2004.11198, 2020.

[52] Alexandre Sablayrolles, Matthijs Douze, Cordelia Schmid, and Herve Jegou. Radioactive data: tracing through training. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 8326–8335. PMLR, 13–18 Jul 2020.

[53] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93, Sep. 2008.

[54] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *CoRR*, abs/1811.05868, 2018.

[55] Yun Shen, Xinlei He, Yufei Han, and Yang Zhang. Model stealing attacks against inductive graph neural networks. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1175–1192, 2022.

[56] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18, 2017.

[57] Jianing Sun, Yingxue Zhang, Wei Guo, Huifeng Guo, Ruiming Tang, Xiuqiang He, Chen Ma, and Mark Coates. Neighbor interaction aware graph convolution networks for recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, page 1289–1298, New York, NY, USA, 2020. Association for Computing Machinery.

[58] Anshuman Suri and David Evans. Formalizing and estimating distribution inference risks. *CoRR*, abs/2109.06024, 2021.

[59] Anshuman Suri, Yifu Lu, Yanjin Chen, and David Evans. Dissecting distribution inference. In *IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, 2023.

[60] Sebastian Szyller, Buse Gul Atli, Samuel Marchal, and N. Asokan. Dawn: Dynamic adversarial watermarking of neural networks. In *Proceedings of the 29th ACM International Conference on Multimedia*, MM '21, page 4417–4425, New York, NY, USA, 2021. Association for Computing Machinery.

[61] Sebastian Szyller, Vasisht Duddu, Tommi Gröndahl, and N. Asokan. Good artists copy, great artists steal: Model extraction attacks against image translation generative adversarial networks. *CoRR*, abs/2104.12623, 2021.

[62] Sebastian Szyller, Rui Zhang, Jian Liu, and N. Asokan. On the robustness of dataset inference, 2022.

[63] Xianfeng Tang, Yandong Li, Yiwei Sun, Huaxiu Yao, Prasenjit Mitra, and Suhang Wang. Transferring robustness for graph neural network against poisoning attacks. In *Proceedings of the 13th international conference on web search and data mining*, pages 600–608, 2020.

[64] Florian Tramer and Dan Boneh. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. In *International Conference on Learning Representations*, 2019.

[65] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In *Proceedings of the 25th*

USENIX Conference on Security Symposium, SEC'16, page 601–618, USA, 2016. USENIX Association.

[66] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. Embedding watermarks into deep neural networks. *CoRR*, abs/1701.04082, 2017.

[67] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

[68] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[69] Eric Wallace, Mitchell Stern, and Dawn Song. Imitation attacks and defenses for black-box machine translation systems. *arXiv preprint arXiv:2004.15015*, 2020.

[70] Si Wang and Chip-Hong Chang. Fingerprinting deep neural networks - a deepfool approach. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2021.

[71] Tianhao Wang and Florian Kerschbaum. RIGA: covert and robust white-box watermarking of deep neural networks. In Jure Leskovec, Marko Grobelnik, Marc Najork, Jie Tang, and Leila Zia, editors, *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, pages 993–1004. ACM / IW3C2, 2021.

[72] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'19, page 165–174, New York, NY, USA, 2019. Association for Computing Machinery.

[73] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *The World Wide Web Conference*, WWW '19, page 2022–2032, New York, NY, USA, 2019. Association for Computing Machinery.

[74] Xiaoqi Wang, Kevin Yen, Yifan Hu, and Han-Wei Shen. Deepgd: A deep learning framework for graph drawing using gnn. *IEEE computer graphics and applications*, 41(5):32–44, 2021.

[75] 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, page 2871–2884, New York, NY, USA, 2022. Association for Computing Machinery.

[76] Bang Wu, Xiangwen Yang, Shirui Pan, and Xingliang Yuan. Adapting membership inference attacks to gnn for graph classification: Approaches and implications. In *2021 IEEE International Conference on Data Mining (ICDM)*, pages 1421–1426, 2021.

[77] Bang Wu, Xiangwen Yang, Shirui Pan, and Xingliang Yuan. Model extraction attacks on graph neural networks: Taxonomy and realisation. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '22, page 337–350, New York, NY, USA, 2022. Association for Computing Machinery.

[78] 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 *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 4816–4823. International Joint Conferences on Artificial Intelligence Organization, 7 2019.

[79] Zhaohan Xi, Ren Pang, Shouling Ji, and Ting Wang. Graph backdoor. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1523–1540. USENIX Association, August 2021.

[80] Jing Xu and Stjepan Picek. Watermarking graph neural networks based on backdoor attacks. *arXiv preprint arXiv:2110.11024*, 2021.

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

[82] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 974–983, New York, NY, USA, 2018. Association for Computing Machinery.

[83] Seongjun Yun, Seoyoon Kim, Junhyun Lee, Jaewoo Kang, and Hyunwoo J Kim. Neo-gnns: Neighborhood overlap-aware graph neural networks for link prediction. *Advances in Neural Information Processing Systems*, 34:13683–13694, 2021.

[84] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *Advances in neural information processing systems*, 31, 2018.

[85] Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. Revisiting graph neural networks for link prediction, 2021.

[86] Xiang Zhang and Marinka Zitnik. Gnnguard: Defending graph neural networks against adversarial attacks. *Advances in neural information processing systems*, 33:9263–9275, 2020.

[87] Yingxue Zhang, S Khan, and Mark Coates. Comparing and detecting adversarial attacks for graph deep learning. In *Proc. Representation Learning on Graphs and Manifolds Workshop, Int. Conf. Learning Representations, New Orleans, LA, USA*, 2019.

[88] Zaixi Zhang, Jinyuan Jia, Binghui Wang, and Neil Zhenqiang Gong. Backdoor attacks to graph neural networks. In *Proceedings of the 26th ACM Symposium on Access Control Models and Technologies*, SACMAT '21, page 15–26, New York, NY, USA, 2021. Association for Computing Machinery.

[89] Zhikun Zhang, Min Chen, Michael Backes, Yun Shen, and Yang Zhang. Inference attacks against graph neural networks. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 4543–4560, Boston, MA, August 2022. USENIX Association.

[90] Jingjing Zhao, Qingyue Hu, Gaoyang Liu, Xiaoqiang Ma, Fei Chen, and Mohammad Mehedi Hassan. Afa: Adversarial fingerprinting authentication for deep neural networks. *Computer Communications*, 150:488–497, 2020.

[91] 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. IEEE, 2021.

[92] Yue Zheng, Si Wang, and Chip-Hong Chang. A dnn fingerprint for non-repudiable model ownership identification and piracy detection. *IEEE Transactions on Information Forensics and Security*, 17:2977–2989, 2022.

[93] Dingyuan Zhu, Ziwei Zhang, Peng Cui, and Wenwu Zhu. Robust graph convolutional networks against adversarial attacks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 1399–1407, New York, NY, USA, 2019. Association for Computing Machinery.

[94] Linghui Zhu, Yiming Li, Xiaojun Jia, Yong Jiang, Shu-Tao Xia, and Xiaochun Cao. Defending against model stealing via verifying embedded external features. In *ICML 2021 Workshop on Adversarial Machine Learning*, 2021.

[95] Michael H. Zhu and Suyog Gupta. To prune, or not to prune: Exploring the efficacy of pruning for model compression, 2018.

[96] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2847–2856, 2018.

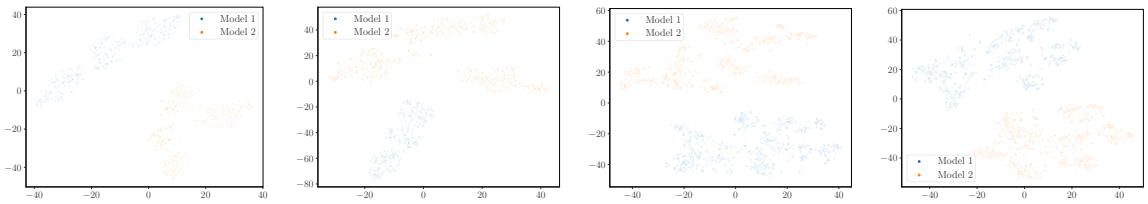
[97] Daniel Zügner, Oliver Borchert, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on graph neural networks: Perturbations and their patterns. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 14(5):1–31, 2020.

APPENDICES

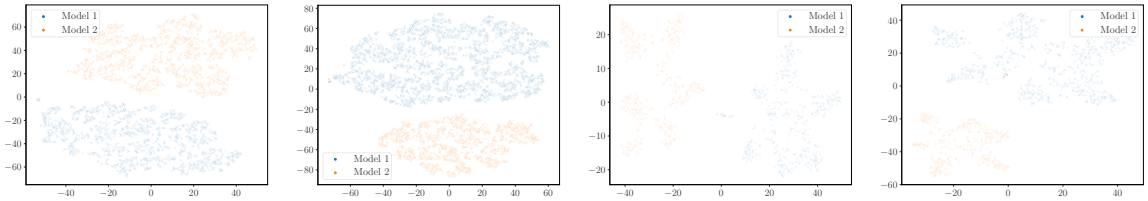
Appendix A

Dataset Ownership Graphs

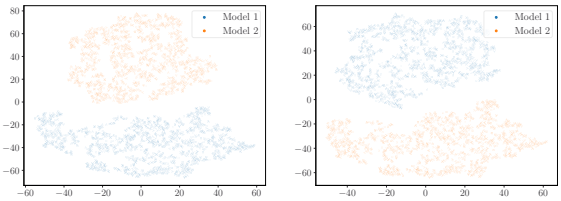
The difference in the two models is most easy to see using CoAuthor since that is the largest dataset. However, the same trend is seen in the rest of the plots, as shown in Figure A.1. Due to space constraints, we only show the graphs for models trained with GAT. The same trend is seen across all architectures.



(a) Diff. Training Data - ACM (b) Same Training Data - ACM (c) Diff. Training Data - Amazon (d) Same Training Data - Amazon



(e) Diff. Training Data - DBLP (f) Same Training Data - DBLP (g) Diff. Training Data - Citeseer (h) Same Training Data - Citeseer



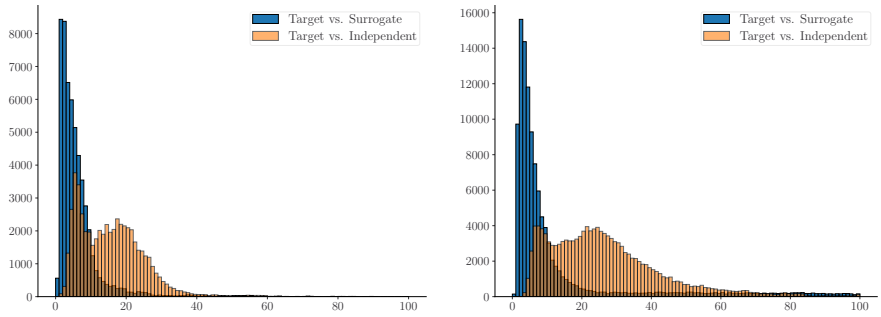
(i) Diff. Training Data - Pubmed (j) Same Training Data - Pubmed

Figure A.1: t-SNE projections of the embeddings from two models trained using the same architecture on are distinguishable for all datasets.

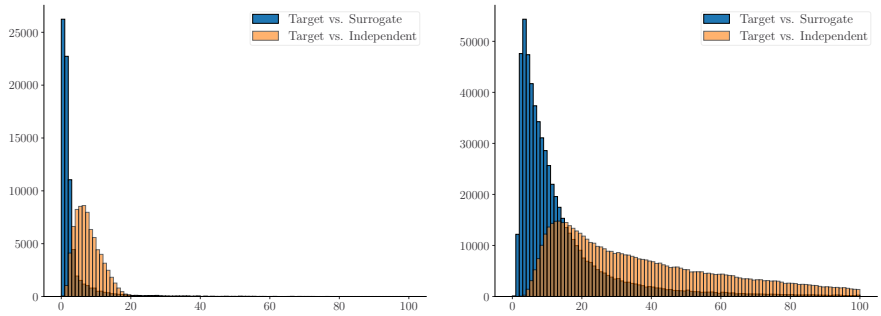
Appendix B

Distance Graphs

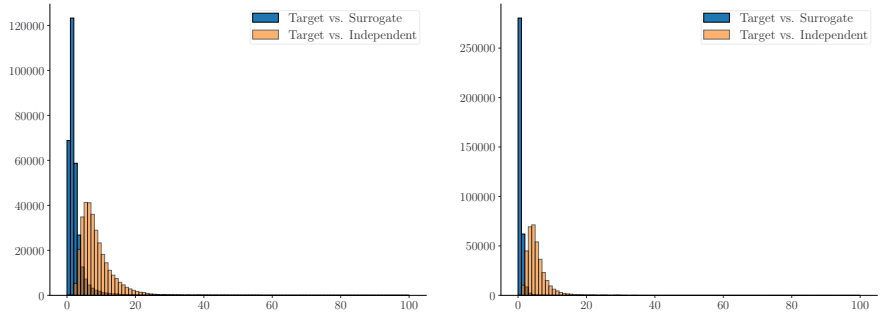
We calculate the Euclidean distance between pairs of embeddings from ($\mathcal{F}_t$, $\mathcal{F}_s$) and ($\mathcal{F}_t$, $\mathcal{F}_i$) and plot the histogram of the distances in Figure B.1. While the distribution of the distances is different, the distributions overlap significantly. Thus, a simple approach based on distance calculation and hypothesis testing did not work.



(a) ACM (b) Amazon



(c) Citeseer (d) CoAuthor



(e) DBLP (f) Pubmed

Figure B.1: Histograms of the Euclidean distances between pairs of embeddings from ($\mathcal{F}_t$, $\mathcal{F}_s$) and ($\mathcal{F}_t$, $\mathcal{F}_i$).