

**DEVELOPING ROBUST MODELS, ALGORITHMS, DATABASES AND TOOLS
WITH APPLICATIONS TO CYBERSECURITY AND HEALTHCARE**

A Dissertation
Presented to
The Academic Faculty

By

Scott Freitas

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in Machine Learning

School of Computational Science and Engineering
Georgia Institute of Technology

December 2021

Copyright © Scott Freitas 2021

**DEVELOPING ROBUST MODELS, ALGORITHMS, DATABASES AND TOOLS
WITH APPLICATIONS TO CYBERSECURITY AND HEALTHCARE**

Approved by:

Duen Horng Chau, Advisor
School of Computational Science &
Engineering
Georgia Institute of Technology

Srijan Kumar
School of Computational Science &
Engineering
Georgia Institute of Technology

Diyi Yang
School of Computational Science &
Engineering
Georgia Institute of Technology

Hanghang Tong
Department of Computer Science
*University of Illinois at Urbana-
Champaign*

B. Aditya Prakash
School of Computational Science &
Engineering
Georgia Institute of Technology

Date Approved: December 7, 2021

To my parents Gary Freitas and Jennifer Morse, to whom I owe everything.

ACKNOWLEDGEMENTS

First and foremost I want to thank my adviser, Duen Horng (Polo) Chau, for his unwavering support and guidance throughout the journey of my PhD. I would not be where I am today if not for your enormous passion, energy and time you have spent to train me to become a researcher. I can never repay you for your endless mentorship and friendship, which has taken many forms over the years—from our weekly meetings, late night paper editing and presentation making, to our endless discussion of ideas and life. Your tireless efforts have forever changed how I see the world.

To my early research mentor and adviser Hanghang Tong, without whom I never would have entered the world of research and graphs. You took a chance on me when I was an undergraduate with no research experience, and through your constant support and mentorship enabled me to become the researcher and person I am today. Without your immeasurable efforts, I never would have been able to achieve early wins, which would forever shape my career and life.

To the countless people who have helped me along this journey—my committee members, Srijan Kumar, Diyi Yang, and B. Aditya Prakash for their invaluable feedback and advice; my close collaborators at IBM Research, Amazon and Microsoft: Jiyong Jang, Frederico Araujo, Teryl Taylor, Hao Zheng, Yanni Lai, Xiaohui Shen, Joshua Neil, Andrew Wicker, Karishma Sanghvi, and Yuxiao Dong for their amazing support and mentorship; my undergraduate research mentors Ross Maciejewski and Yezhou Yang for their invaluable advice and support; my dear friend Rahul Duggal—I could not have made it through this program without you. Your support, and our conversations about life and research enabled me to keep going even when I thought I could not. To all my friends and colleagues in the PoloClub of Data Science: Fred Hohman, Haekyu Park, Nilaksh Das, Jay Wang and Austin Wright I am lucky to have had your friendship and advice. Also to Kevin Allix and AndroZoo colleagues for generously allowing us to use their data in our research.

Lastly, I want to thank you parents Gary Freitas and Jennifer Morse for their unconditional support throughout this long and arduous journey—you are the reason I am here today.

TABLE OF CONTENTS

Acknowledgments	iv
List of Tables	x
List of Figures	xiii
Summary	xix
Chapter 1: Introduction	1
1.1 Thesis Overview	1
1.1.1 Part I: Robust Tools	1
1.1.2 Part II: Algorithms	4
1.1.3 Part III: Databases	6
1.1.4 Part IV: Robust Models	7
1.2 Thesis Statement	10
1.3 Research Contributions	10
1.4 Impact	11
 I Robust Tools	 13
Chapter 2: Graph Vulnerability and Robustness: A Survey	15
2.1 Introduction	15
2.1.1 Contributions	16
2.1.2 Survey Methodology & Summarization Process	18
2.1.3 Related Surveys	18
2.1.4 Survey Organization.	20
2.2 Robustness Measures	20
2.2.1 Measures Based on Graph Connectivity	22
2.2.2 Measures Based on Adjacency Matrix Spectrum	28
2.2.3 Measures Based on Laplacian Matrix Spectrum	32
2.2.4 Comparing Robustness Measures	34
2.2.5 Selecting a Robustness Measure	35
2.3 Failures and Targeted Attacks	37
2.3.1 Graph Models	37
2.3.2 Isolated & Cascading Failures	39
2.3.3 Targeted Attacks	39

2.3.4	Comparison to Other Targeted Attacks	42
2.4	Network Defense	42
2.4.1	Measure Independent Heuristics	43
2.4.2	Optimization Based Techniques	44
2.4.3	Selecting a Defense Method	45
2.5	Research Directions & Open Problems	46
2.5.1	Guidelines for Selecting & Developing Measures	46
2.5.2	Furthering Interpretability	47
2.5.3	Studying Robustness in New Domains	47
2.5.4	Bridging Graph Robustness & Adversarial ML	48
2.6	Conclusion	48
Chapter 3: Evaluating Graph Vulnerability and Robustness using TIGER . . .		50
3.1	Introduction	50
3.1.1	Contributions	52
3.2	TIGER Robustness Measures	53
3.2.1	Example Measures	54
3.2.2	Measure Implementation & Evaluation	55
3.2.3	Running Robustness Measures in TIGER	56
3.3	TIGER Attacks	57
3.3.1	Attack Strategies	57
3.3.2	Comparing Strategies	58
3.3.3	Running Network Attacks in TIGER	58
3.4	TIGER Defenses	59
3.4.1	Defense Strategies	59
3.4.2	Comparing Strategies	60
3.4.3	Running Network Defenses in TIGER	60
3.5	TIGER Simulation Tools	62
3.5.1	Cascading Failures	62
3.5.2	Running Cascading Failures in TIGER	64
3.5.3	Dissemination of Network Entities	65
3.5.4	Running Entity Dissemination in TIGER	65
3.6	Conclusion	67

II Robust Algorithms 68

Chapter 4: D²M: Dynamic Defense and Modeling of Adversarial Movement in Networks		70
4.1	Introduction	70
4.2	Background and Our Differences	72
4.2.1	Detecting Lateral Attacks	73
4.2.2	Graph Mining & Network Security	73
4.3	Authentication Graph	74
4.3.1	Building Graph Structure	74

4.3.2	Integrating Domain Knowledge	74
4.4	Formulating the Research Problems	76
4.5	D ² M: Lateral Attack Modeling	76
4.5.1	Lateral Attack Overview	76
4.5.2	Lateral Attack Strategies	78
4.5.3	Lateral Attack Algorithm	79
4.5.4	Analysis of Lateral Attack Algorithm	80
4.6	D ² M: Lateral Attack Vulnerability	80
4.7	D ² M: Lateral Attack Defense	82
4.7.1	Defense Strategies	82
4.7.2	Analysis of Defense Strategies	84
4.8	Experiments	84
4.8.1	Experimental Setup	84
4.8.2	Network Vulnerability Analysis	85
4.8.3	Defense Strategy Analysis	85
4.9	Conclusion	86

III Robust Databases 88

Chapter 5: A Large-Scale Database for Graph Representation Learning 90

5.1	Introduction	90
5.2	Properties of MalNet	92
5.2.1	Graph Representation Learning Databases: Advancing the State-of-the-Art	94
5.3	Constructing MalNet	96
5.3.1	Collecting Candidate Graphs	96
5.3.2	Processing the Graphs	97
5.3.3	MalNet-Tiny	97
5.3.4	Online Exploration of the Data	98
5.4	MalNet for New Research & Discoveries	98
5.4.1	Graph Representation Techniques	98
5.4.2	Enabling New Discoveries	100
5.4.3	Enabling New Research Directions	101
5.5	Conclusion	103

Chapter 6: A Large-Scale Image Database of Malicious Software 105

6.1	Introduction	105
6.2	Properties of MalNet-Image	107
6.3	MalNet-Image: Advancing the State-of-the-Art	108
6.3.1	Constructing MalNet-Image	110
6.3.2	Interactive Visual Explorer for MALNET-IMAGE	112
6.4	MalNet-Image Applications	113
6.4.1	Application 1: Benchmarking Techniques	113
6.4.2	Application 2: Malware Detection	116

6.4.3	Application 3: Malware Classification	117
6.4.4	Enabling New Research Directions	118
6.5	Conclusion	119

IV Robust Models 121

Chapter 7: UnMask: Adversarial Detection and Defense Through Robust Feature Alignment

	ture Alignment	123
7.1	Introduction	123
7.1.1	Contributions	124
7.2	Background and Related Work	126
7.2.1	Adversarial Attacks	126
7.2.2	Adversarial Defense & Detection	127
7.3	UNMASK: Detection and Defense Framework	128
7.3.1	Aligning Robust Features with Human Intuition	128
7.3.2	Robust Features For Detection and Defense	130
7.4	Evaluation	132
7.4.1	Experiment Setup	132
7.4.2	Evaluating UnMask Defense and Detection	135
7.5	Conclusion	138

Chapter 8: REST: Robust and Efficient Neural Networks for Sleep Monitoring in the Wild

	in the Wild	142
8.1	Introduction	142
8.1.1	Contributions	144
8.2	Related Work	145
8.2.1	Sleep-Stage Prediction	145
8.2.2	Noise & Adversarial Robustness	146
8.2.3	Model Compression	146
8.3	REST: Noise-Robust & Efficient Models	147
8.3.1	Overview	147
8.3.2	Adversarial Training	147
8.3.3	Spectral Regularizer	148
8.3.4	Sparsity Regularizer & REST Loss Function	150
8.4	Experiments	151
8.4.1	Datasets	151
8.4.2	Model Architecture and Configurations	153
8.4.3	Evaluation Metrics	154
8.4.4	Hyperparameter Selection	155
8.4.5	Noise Robustness	157
8.4.6	Model Efficiency	161
8.5	Conclusion	162

V	Conclusions	163
Chapter 9:	Conclusion and Future Directions	164
9.1	Research Contributions	164
9.2	Impact	165
9.3	Future Directions	166
9.3.1	Advancing Vision Based Cybersecurity Research	166
9.3.2	Advancing Graph Representation Learning Research	167
9.3.3	Robust Tools and Algorithms	168
References		200

LIST OF TABLES

1.1	The publications mapped to the thesis outline.	2
2.1	Relevant venues in order of journals, conferences, workshops, and preprints. Within each category, we order the venues based on the most recently available impact factors reported officially (e.g., venues’ websites).	19
2.2	Summary of works studied in this survey, each row is one work. Columns are grouped into one of three categories—robustness measures, attacks and defenses—corresponding to primary chapter sections (except “where”). In addition, we divide the robustness measure columns into three categories based on whether it uses the graph, adjacency matrix, or Laplacian matrix, from left to right, respectively (using dashed lines)	22
2.3	Symbols and Definition Tables. We divide symbols and definitions based on whether it corresponds to use with the graph, adjacency matrix or Laplacian matrix. From left to right, symbol and definition tables for the graph, adjacency matrix and Laplacian matrix.	23
2.4	Comparison of robustness measures. Measures are grouped based on whether they use the <i>graph</i> , <i>adjacency</i> or <i>Laplacian</i> matrix. For each measure, we briefly describe it’s application to measuring network robustness.	35
4.1	Symbols and Definition	73
4.2	Graph Statistics. ρ : graph density, C : average clustering coefficient, δ_{avg} : mean node out-degree.	85
4.3	Vulnerability Statistics. Statistics excluded for G_{lanl} strategies RE and DE in h_3 as computation exceeded budget (Sect. 4.8.1).	86
5.1	Descriptive statistics for 10 largest graph types. See Table 5.4 for all graph statistics.	92

5.2	Comparison of MALNET-GRAPH properties with common graph classification datasets. MALNET-GRAPH offers over 1.2 million graphs averaging $15k$ nodes and $35k$ edges with a hierarchical class structure containing 47 types and 696 families. This makes MALNET-GRAPH the largest public graph database constructed to date, offering $105\times$ more graphs, $39\times$ larger graphs on average, and $63\times$ more classes compared to the popular REDDIT-12K database. CC is the clustering coefficient.	95
5.3	Comparison of macro-F1, precision and recall scores achieved by 7 methods at the <i>type</i> (low diversity, with 47 classes) and <i>family</i> (high diversity, with 696 classes) and <i>tiny</i> (5k graphs across 5 balanced classes) classification levels. Comparing methods across <i>type</i> and <i>family</i> , the classification task becomes increasingly difficult as diversity and data imbalance increase.	101
5.4	Descriptive statistics for each graph type in MALNET-GRAPH.	104
6.1	MALNET-GRAPH has 1.2M images across a hierarchy of 47 types and 696 families.	106
6.2	The number of images and families in each type of malware in MALNET-GRAPH.	109
6.3	We evaluate the performance of 3 popular architectures—ResNet, DenseNet and MobileNetV2—on its macro-F1, macro-precision, and macro-recall. Model performance is similar across architectures, while model size (parameters) and computational cost (MFlops) varies widely. As a result, we conduct all additional experiments using a ResNet18 model as it provides a strong balance between performance and training time.	114
7.1	Class-Feature Matrix. Top: dots mark classes' features. Bottom: four class sets with varying levels of feature overlap. Features <i>vehicle</i> and <i>coach</i> have sub-features not listed here due to space (see Github repository).	139
7.2	Number of images used to train and evaluate models K , M and defense framework D . We train K on PASCAL-Part dataset, and model M on PASCAL VOC 2010 plus a subset of ImageNet. Four <i>class sets</i> are investigated in the evaluation, with varying classes and feature overlap. We evaluate model M and defense framework D on Flickr.	140

7.3	Number of images used to evaluate the detection capability of UNMASK. Only images that are successfully attacked are used for evaluation (combined with their benign counterparts), thus the variations in numbers. We report values for PGD and MIA with $\epsilon=16$, respectively. Numbers are similar for $\epsilon=8$	140
7.4	Accuracies in countering 4 strong attacks at 2 strength levels (PGD- L_∞ , PGD- L_2 , MIA- L_∞ , MIA- L_2), using 2 CNN architectures as unprotected model M across 4 class sets. UNMASK (“UM”) provides significantly better protection than adversarial training (“AT”), 31.18% on average. “None” means no defense.	141
8.1	Dataset summary outlining the number of 30 second EEG recordings belonging to each sleep stage class.	151
8.2	Line search results for identifying optimal c_g on Sleep-EDF and SHHS datasets. Macro-F1 is abbreviated F1 in table; average macro-F1 is the mean of all macro-F1 scores. We select $c_g=0.2$ for both datasets as it represents a good trade-off between benign and Gaussian macro-F1.	156
8.3	Grid search results for λ_o and λ_g on Sleep-EDF dataset. Macro-F1 is abbreviated as F1 in table; average macro-F1 is the mean of all macro-F1 scores. We select λ_o and λ_g with highest average macro-F1 score.	156
8.4	Meta Analysis: Comparison of macro-F1 scores achieved by each model. The models are evaluated on Sleep-EDF and SHHS datasets with three types and strengths of noise corruption. We bold the compressed model with the best performance (averaged over 3 runs) and report the standard deviation of each model next to the macro-F1 score. REST performs better in <i>all</i> noise test measurements.	157
8.5	Comparison on model size and the FLOPS required to score a single night of EEG recordings. REST models are significantly smaller and comparable in size/compute to baselines.	161

LIST OF FIGURES

1.1	A visual overview of the robustness work surveyed.	3
1.2	TIGER provides a number of important tools for graph vulnerability and robustness research, including graph robustness measures, attack strategies, defense techniques and simulation models. Here, a TIGER user is visualizing a computer virus simulation that follows the SIS infection model (effective strength $s = 3.21$) on the <i>Oregon-1 Autonomous System</i> network [4]. Top: defending only 5 nodes with <i>Netshield</i> [5], the number of infected entities is reduced to nearly zero. Bottom: without any defense, the virus remains endemic.	4
1.3	D ² M framework: 1. Builds an authentication graph from device authentication history; 2. Allows security analysts to test different attack strategies to study network vulnerability; 3. Identifies at-risk machines to monitor, preempting lateral attacks.	5
1.4	MALNET-GRAPH: Advancing State-of-the-Art Graph Databases. MALNET-GRAPH contains 1, 262, 024 function call graphs averaging 17, 242 nodes and 39, 043 edges per graph, across a hierarchy of 47 types and 696 families of malware.	7
1.5	UNMASK combats adversarial attacks (in red) through extracting <i>robust features</i> from an image (“Bicycle” at top), and comparing them to expected features of the classification (“Bird” at bottom) from the unprotected model. Low feature overlap signals an attack.	9
1.6	REST Overview: (from left) When a noisy EEG signal belonging to the REM (rapid eye movement) sleep stage enters a traditional neural network which is vulnerable to noise, it gets wrongly classified as a Wake sleep stage. On the other hand, the same signal is correctly classified as the REM sleep stage by the REST model which is both robust and sparse. (From right) REST is a three step process involving (1) training the model with adversarial training, spectral regularization and sparsity regularization (2) pruning the model and (3) re-training the compact model.	9

2.1	A visual overview of the work surveyed . §2 summarizes and compares 18 graph robustness. §3 overviews methods of network failure and attack. §4 summarizes network defense techniques across a variety of graph topologies and attack vectors.	17
3.1	TIGER provides a number of important tools for graph vulnerability and robustness research, including graph robustness measures, attack strategies, defense techniques and simulation models. Here, a TIGER user is visualizing a computer virus simulation that follows the SIS infection model (effective strength $s = 3.21$) on the <i>Oregon-1 Autonomous System</i> network [4]. Top: without any defense, the virus remains endemic. Bottom: defending only 5 nodes with <i>Netshield</i> [5], the number of infected entities is reduced to nearly zero.	51
3.2	Error of 5 fast, approximate robustness measures supported by TIGER. Parameter k represents the trade-off between speed (low k) and precision (high k). To measure approximation efficacy, we vary $k \in [5, 300]$ in increments of 10 and measure the error between the approximate and original measure averaged over 30 runs on a clustered scale-free graph with 300 nodes.	53
3.3	TIGER simulation of an RD node attack on the KY-2 water distribution network. Step 0: network starts under normal conditions; at each step a node is removed by the attacker (red nodes). Step 13, 22 & 27: after removing only a few of the 814 nodes, the network splits into two and three and four disconnected regions, respectively.	56
3.4	Efficacy of 5 edge attacks (left) and 5 node attacks (right) on the KY-2 water distribution network. The most effective attack (RB) disconnects approximately 50% of the network with less than 30 removed edges (or nodes).	58
3.5	Comparing ability of 5 edge defenses to improve KY-2 network robustness after removing 30 nodes via RB attack. Edge addition performs the best, with random edge rewiring performing the worst.	61
3.6	Effect of network redundancy r on the US power grid where 4 nodes are overloaded using ID. When $r \geq 50\%$ the network is able to redistribute the increased load.	62

3.7	TIGER cascading failure simulation on the US power grid network when 4 nodes are overloaded according to the ID attack strategy. Time step 1: shows the network under normal conditions. Time step 50: we observe a series of failures originating from the bottom of the network. Time step 70: most of the network has collapsed.	63
3.8	SIS simulation with 5 virus strengths on the Oregon-1 Autonomous System network. No defense (left), Netshield defense (right).	66
4.1	Our D^2M framework: 1. Builds an authentication graph from device authentication history; 2. Allows security analysts to test different attack strategies to study network vulnerability; 3. Identifies at-risk machines to monitor, preempting lateral attacks.	71
4.2	Attack path generated by D^2M . 1. Network is penetrated; 2-4. Attacker explores the network and escalates privileges; 5. Attacker compromises the domain controller, gaining control of the network.	77
4.3	Each defense strategy is compared on three graphs and attack strategies, where ANOMALYSHIELD performs well across a majority of application scenarios.	87
5.1	MALNET-GRAPH has 1.2M graphs averaging 15k nodes and 35k edges per graph.	91
5.2	Example of the graph type “worm” and its 7 families.	93
5.3	FCG from the <i>Banker++Trojan</i> type, and <i>Acecard</i> family. Nodes represent functions and edges indicate inter-procedural calls. Highlighted in blue is <i>one</i> potential execution path.	94
5.4	Class-wise comparison of model predictions where a darker cell represents a higher F1 score. We observe that certain classes are more challenging to classify than others.	102
6.1	<i>Type</i> and <i>family</i> labels have imbalance ratios of $7,827\times$ and $16,901\times$, respectively.	108
6.2	Left: Android DEX file structure, composed of three major components—(1) header, (2) ids, and (3) data. Right: binary image representation of the DEX file.	110

6.3	Images of two malware types with different “texture”. Left: the Trojan image is more “fine-grained”. Right: the Adware image is more “coarse”. Malware images belonging to the same type or family often appear visually similar in layout and texture, whereas images across types and families contain noticeable differences in layout and texture.	111
6.4	MALNET-IMAGE EXPLORER. An exploration panel on the left allows users to select from the available images types and families. Users can then visually the image on the right.	112
6.5	Malware detection ROC curve with an AUC of 0.94, demonstrating the potential of binary images as an effective form of malware detection. . . .	116
6.6	Model attention patterns across 4 types of malware (each with 2 images). Ransom++Trojan: narrowly focused on thin region of data section. Benign: wide range of attention across data section. Adware: attention on circular bytecode “hotspots”. Monitor: focus on “empty” black region of data section.	117
6.7	Malware classification results using confusion matrix heatmap (classes in descending order of size). We analyze type level classification performance, where a dark diagonal indicates strong performance, and a dark off-diagonal indicates poor performance. Each square in the diagonal indicates the percent of examples correctly classified for a particular malware type; and each off-diagonal entry indicates the percent of incorrectly classified examples for a particular type.	120
7.1	UNMASK combats adversarial attacks (in red) through extracting <i>robust features</i> from an image (“Bicycle” at top), and comparing them to expected features of the classification (“Bird” at bottom) from the unprotected model. Low feature overlap signals an attack. UNMASK rectifies misclassification using the image’s extracted features. Our approach <i>detects</i> 96.75% of gray-box attacks (at 9.66% false positive rate) and <i>defends</i> the model by correctly classifying up to 93% of adversarial images crafted by Projected Gradient Descent (PGD).	124
7.2	Line search for adversarial training parameter ϵ on validation data. We select $\epsilon = 4$, since it provides the best performance on most attacks.	133
7.3	Detailed bar chart describing the performance of ϵ across all attack vectors. We select $\epsilon = 4$, since it provides the best performance on most attacks. . . .	134

7.4	Accuracies (in %) for each class set averaged across all attack vectors, strengths, and models from Table 7.4. On average, UNMASK (UM) performs 31.18% better than adversarial training (AT) and 74.44% than no defense (None).	136
7.5	UNMASK’s effectiveness in detecting 4 strong attacks at two strength levels. UNMASK’s protection may not be affected strictly based on the number of classes. Rather, an important factor is the <i>feature overlap</i> among classes. UNMASK provides better detection when there are 5 classes (dark orange; 23.53% overlap) than when there are 3 (light blue; 50% overlap). Keeping the number of classes constant and varying their feature overlap also supports our observation about the role of feature overlap (e.g., CS3a at 6.89% vs. CS3b at 50%). Dotted line indicates random guessing.	137
8.1	Top: we generate hypnograms for a patient in the SHHS test set. In the presence of Gaussian noise, our REST -generated hypnogram closely matches the contours of the expert -scored hypnogram. Hypnogram generated by a state-of-the-art (SOTA) model by Sors et al. [316] is considerably worse. Bottom: we measure energy consumed (in Joules) and inference time (in seconds) on a smartphone to score one night of EEG recordings. REST is 9X more energy efficient and 6X faster than the SOTA model.	143
8.2	REST Overview: (from left) When a noisy EEG signal belonging to the REM (rapid eye movement) sleep stage enters a traditional neural network which is vulnerable to noise, it gets wrongly classified as a Wake sleep stage. On the other hand, the same signal is correctly classified as the REM sleep stage by the REST model which is both robust and sparse. (From right) REST is a three step process involving (1) training the model with adversarial training, spectral regularization and sparsity regularization (2) pruning the model and (3) re-training the compact model.	144
8.3	Line search results for ϵ on Sleep-EDF and SHHS datasets. We select $\epsilon=10$, since it provides the best average macro-F1 score on both datasets.	155
8.4	Meso Analysis: Class-wise comparison of model predictions. The models are evaluated over the SHHS test set perturbed with different noise types. In each confusion matrix, rows are ground-truth classes while columns are predicted classes. The intensity of a cell is obtained by normalizing the score with respect to the class membership. When a cell has a value of 1 (dark blue, or dark green) the model predicts every example correctly, the opposite occurs at 0 (white). A model that is performing well would have a dark diagonal and light off-diagonal. REST has the darkest cells along the diagonal on both datasets.	159

8.5	Granular Analysis: Comparison of the overnight hypnograms obtained for a patient in the SHHS test set. The hypnograms are generated using the Sors (left) and REST (right) models in the presence of increasing strengths of Gaussian noise. When no noise is present (top row), both models perform well, closely matching the ground truth (bottom row). However, with increasing noise, Sors performance rapidly degrades, while REST continues to generate accurate hypnograms.	160
8.6	Time and energy consumption for scoring a single night of EEG recordings. REST(A+S) is significantly faster and more energy efficient than the state-of-the-art Sors model. Evaluations were done on a Pixel 2 smartphone. . . .	160

SUMMARY

As society and technology becomes increasingly interconnected, so does the threat landscape. Once isolated threats now pose serious concerns to highly interdependent systems, highlighting the fundamental need for robust machine learning. This dissertation contributes novel *tools, algorithms, databases and models—through the lens of robust machine learning*—in a research effort to solve large-scale societal problems affecting millions of people in the areas of cybersecurity and healthcare.

(1) **Tools:** We develop TIGER, the first comprehensive graph robustness toolbox; and our ROBUSTNESS SURVEY identifies critical yet missing areas of graph robustness research.

(2) **Algorithms:** Our survey and toolbox reveal existing work has overlooked lateral attacks on computer authentication networks. We develop D²M, the first algorithmic framework to quantify and mitigate network vulnerability to lateral attacks by modeling lateral attack movement from a graph theoretic perspective.

(3) **Databases:** To prevent lateral attacks altogether, we develop MALNET-GRAPH, the worlds largest cybersecurity graph database—containing over 1.2M graphs across 696 classes—and show the first large-scale results demonstrating the effectiveness of malware detection through a graph medium. We extend MALNET-GRAPH by constructing the largest binary-image cybersecurity database—containing 1.2M images, $133\times$ more images than the only other public database—enabling new discoveries in malware detection and classification research restricted to a few industry labs (MALNET-IMAGE).

(4) **Models:** To protect systems from adversarial attacks, we develop UNMASK, the first model that flags semantic incoherence in computer vision systems, which *detects* up to 96.75% of attacks, and *defends* the model by correctly classifying up to 93% of attacks. Inspired by UNMASK’s ability to protect computer visions systems from adversarial attack, we develop REST, which creates noise robust models through a novel combination of *adversarial training, spectral regularization and sparsity regularization*. In the presence of noise, our method improves state-of-the-art sleep stage scoring by 71%—allowing us to diagnose sleep disorders earlier on and in the home environment—while using $19\times$ less parameters and $15\times$ less MFLOPS.

Our work has made significant impact to industry and society: the UNMASK framework laid the foundation for a multi-million dollar DARPA GARD award; the TIGER toolbox for graph robustness analysis is a part of the Nvidia Data Science Teaching Kit, available to educators around the world; we released MALNET, the world’s largest graph classification database with 1.2M graphs; and the D²M framework has had major impact to Microsoft products, inspiring changes to the product’s approach to lateral attack detection.

CHAPTER 1

INTRODUCTION

While the advancing era of web connected devices has resulted in many technological breakthroughs, the interconnected nature of these devices has created a demand for robust machine learning that is resilient to both natural threats and targeted actors. In its recent 2020 report, the U.S. National Science and Technology Council wrote a report [1] outlined how AI is changing the landscape of cyber warfare, detailing how cyber systems will automatically carry out attacks and defend cyberspace with the assistance of human-in-the-loop operators. Through my diverse research experience at government research laboratories, private defense contractors, Microsoft’s Advanced Threat Protection team, and Amazon’s Fraud Detection and Risk Transaction team, it is clear that artificial intelligence is transforming cybersecurity and healthcare across the public and private sector. The question is, how do we robustify artificial intelligence to defend against this vast threat landscape? This thesis advances the state-of-the-art through the development of robust models, algorithms, databases and tools to solve large-scale societal problems in cybersecurity and healthcare, by addressing real-world problems affecting millions of people. The publications and work from this research are listed in Table 1.1

1.1 Thesis Overview

We provide an overview of the thesis, listing the problems we address and presenting a summary of our contributions. Our research groups into four interrelated topics, which form the main thrusts of the thesis.


1.1.1 Part I: Robust Tools


Attack campaigns from criminal organizations and nation state actors are quickly becoming one of the most powerful forms of disruption. In 2016 alone, malicious cyber activity cost the U.S. economy between \$57 and \$109 billion [2]. These cyber-attacks are often highly sophisticated, targeting governments and large-scale enterprises to interrupt critical services and steal intellectual property [3]. This has led to a rallying cry for new cybersecurity defense systems that can democratize cybersecurity knowledge and tools have been scattered across are disparate technical fields, or in the possession of a few industry labs.

Graph Vulnerability and Robustness: A Survey (Chapter 2). We present a survey on


Table 1.1: The publications mapped to the thesis outline.

Part I: Robust Tools


 Graph Vulnerability and Robustness: A Survey. Scott Freitas, Diyi Yang, Srijan Kumar, Hanghang Tong, Duen Horng Chau. **[Under review]** *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2021. <https://arxiv.org/pdf/2105.00419.pdf> (Chapter 2)


 Evaluating Graph Vulnerability and Robustness using TIGER. Scott Freitas, Diyi Yang, Srijan Kumar, Hanghang Tong, Duen Horng Chau. *ACM International Conference on Information and Knowledge Management (CIKM)*, 2021. <https://arxiv.org/abs/2006.05648> (Chapter 3)

Part II: Robust Algorithms


 D²M: Dynamic Defense and Modeling of Adversarial Movement in Networks. Scott Freitas, Andrew Wicker, Duen Horng Chau, Joshua Neil. *SIAM International Conference on Data Mining (SDM)*, 2020. <https://arxiv.org/abs/2001.11108> (Chapter 4)


Part III: Robust Databases

 A Large-Scale Database for Graph Representation Learning. Scott Freitas, Yuxiao Dong, Joshua Neil, Duen Horng Chau. *Neural Information Processing Systems (NIPS) Datasets and Benchmarks*, 2021. <https://arxiv.org/abs/2011.07682> (Chapter 5)

 A Large-Scale Image Database of Malicious Software. Scott Freitas, Rahul Duggal, Duen Horng Chau. **[Submitting to]** *Knowledge Discovery and Data Mining (KDD)*, 2022. <https://arxiv.org/abs/2102.01072> (Chapter 6)

Part IV: Robust Models

 UnMask: Adversarial Detection and Defense Through Robust Feature Alignment. Scott Freitas, Shang-Tse Chen, Zijie J. Wang, Duen Horng Chau. *IEEE International Conference on Big Data (Big Data)*, 2020. <https://arxiv.org/abs/2002.09576> (Chapter 7)

 REST: Robust and Efficient Neural Networks for Sleep Monitoring in the Wild. Rahul Duggal*, Scott Freitas*, Cao Xiao, Duen Horng Chau, Jimeng Sun. *Proceedings of The Web Conference (WWW)*, 2020. * Both authors contributed equally to this research. <https://arxiv.org/abs/2001.11363> (Chapter 8)

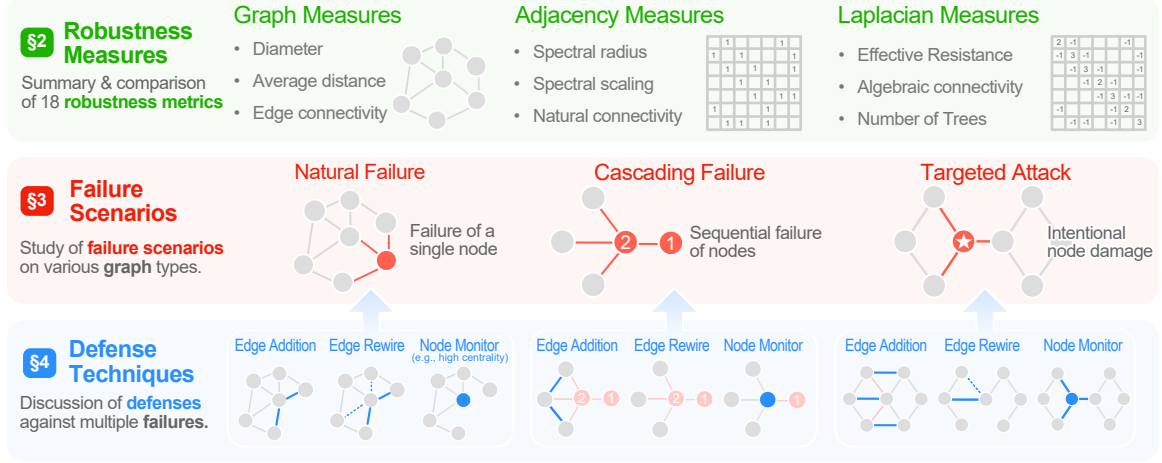


Figure 1.1: A visual overview of the robustness work surveyed.

the role of network robustness as a critical tool in the characterization and understanding of complex interconnected systems such as infrastructure, communication and social networks. We find that answers to key robustness questions are currently scattered across multiple scientific fields and numerous papers. In this survey, we distill key findings across numerous domains and provide researchers crucial access to important information by—(1) summarizing and comparing recent and classical graph robustness measures; (2) exploring which robustness measures are most applicable to different categories of networks (e.g., social, infrastructure); (3) reviewing common network attack strategies, and summarizing which attacks are most effective across different network topologies; and (4) extensive discussion on selecting defense techniques to mitigate attacks across a variety of networks (see Figure 1.1 for a visual overview). This survey guides researchers and practitioners in navigating the expansive field of network robustness, while summarizing answers to key questions.

Evaluating Graph Vulnerability and Robustness using TIGER (Chapter 3). Having extensively surveyed the field of graph vulnerability and robustness, we find that the cross-disciplinary nature of graph robustness research often means that important discoveries made in one field are not quickly disseminated to others, hindering reproducibility and examination of existing work, development of new research, and dissemination of new ideas. Unfortunately, no comprehensive open-source toolbox currently exists to assist researchers and practitioners in this important topic. We believe a unified and easy-to-use software framework is key to standardizing the study of network robustness, helping accelerate reproducible research and dissemination of ideas. Through analyzing and understanding the robustness of these networks we can: (1) quantify network vulnerability and robustness, (2) augment a network’s structure to resist attacks and recover from failure, and (3) con-

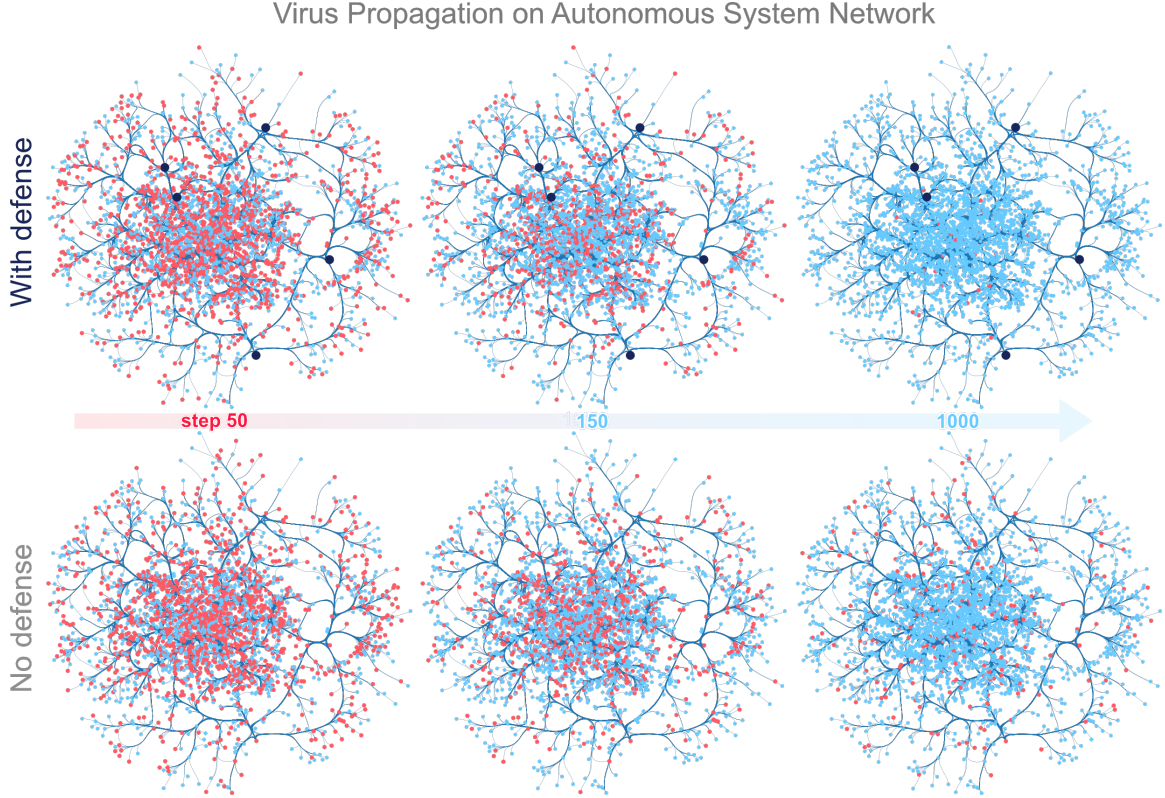


Figure 1.2: TIGER provides a number of important tools for graph vulnerability and robustness research, including graph robustness measures, attack strategies, defense techniques and simulation models. Here, a TIGER user is visualizing a computer virus simulation that follows the SIS infection model (effective strength $s = 3.21$) on the *Oregon-1 Autonomous System* network [4]. Top: defending only 5 nodes with *Netshield* [5], the number of infected entities is reduced to nearly zero. Bottom: without any defense, the virus remains endemic.

trol the dissemination of entities on the network (e.g., viruses, propaganda). We contribute TIGER, an open-sourced Python toolbox to address these challenges. TIGER contains 22 graph robustness measures with both original and fast approximate versions; 17 failure and attack strategies; 15 heuristic and optimization based defense techniques; and 4 simulation tools (see Figure 1.2 for one type of simulation and defense tool available in TIGER). By democratizing the tools required to study network robustness, our goal is to assist researchers and practitioners in analyzing their own networks; and facilitate the development of new research in the field.

1.1.2 Part II: Algorithms

Through our survey and the development of the TIGER toolbox, we find that network robustness research primarily focuses on the development of algorithms to study social networks and infrastructure networks, but fails to address important issues in the domain

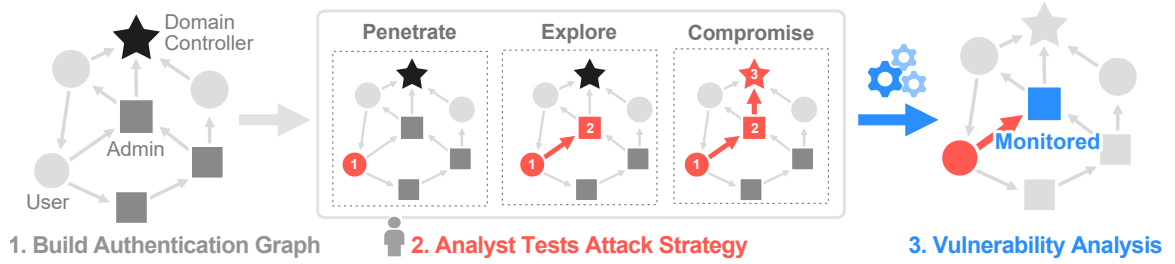


Figure 1.3: D²M framework: **1.** Builds an authentication graph from device authentication history; **2.** Allows security analysts to test different attack strategies to study network vulnerability; **3.** Identifies at-risk machines to monitor, preempting lateral attacks.

of cybersecurity. In particular, as a first step we chose to address this problem by studying robustness of authentication graphs in enterprise networks to lateral attacks (D²M). Working with researchers, engineers and threat hunters in the Microsoft Advanced Threat Protection (ATP) group, we developed D²M, the first graph-theoretic framework that systematically quantifies network vulnerability to lateral attack and identifies at-risk devices (see Figure 1.3). This is a high impact problem, since once an attacker has compromised a single credential for an enterprise machine, the whole network becomes vulnerable to lateral attack movements [6], allowing the adversary to eventually gain control of the network (i.e., escalating privileges via credential stealing [7]).

D²M: Dynamic Defense and Modeling of Adversarial Movement in Networks (Chapter 4). Our survey and toolbox reveal existing work has overlooked lateral attacks on computer authentication networks. We develop *D²M*, the first algorithmic framework to quantify and mitigate network vulnerability to lateral attacks by modeling lateral attack movement from a graph theoretic perspective. We formulate a novel Monte-Carlo method that calculates network robustness to lateral attacks as a probabilistic function of the network topology, and then develop a suite of five fast graph mining techniques to identify enterprise machines at risk to lateral attacks. Despite the prevalence of lateral attacks, observing and analyzing them is challenging for multiple reasons: (1) lateral attacks are still relatively sparse compared to the unsuccessful attack; (2) attack ground-truth is hard to ascertain, and generally partially uncovered through investigation; and (3) due to the fact that the adversary already has a valid credential for the network, attackers can operate as a legitimate user. While real attack data does exist—due to the above challenges, it is rarely fully visible, or accessible, making the study of a “complete” attack highly problematic. D²M has led to major impact to the Microsoft Defender Advanced Threat Protection product, inspiring changes to the product’s approach to detect and prevent lateral movement, well known as one of the most challenging areas of post-breach detection.

1.1.3 Part III: Databases

To prevent lateral attacks altogether (Part 1.1.2), we create two new cybersecurity databases to enable the development of next-generation malware detection models. We develop the worlds largest cybersecurity graph database—containing over 1.2 million graphs across 696 classes—and show the first large-scale results demonstrating the effectiveness of malware detection through a graph medium (MALNET-GRAPH). We then expand upon MALNET-GRAPH by constructing the largest binary-image cybersecurity database—containing 1.2 million images, $24\times$ more images than the only other public database—enabling new discoveries in malware detection and classification research previously restricted to a few industry labs (MALNET-IMAGE).

A Large-Scale Database for Graph Representation Learning (Chapter 5). A majority of malware samples are *polymorphic* in nature, meaning that subtle source code changes in the original malware variant can result in significantly different compiled code (e.g., instruction reordering, branch inversion, register allocation) [8, 9]. Cybercriminals frequently take advantage of this to evade signature based detection, a predominant form of malware detection [10]. Fortunately, these subtle source code changes have minimal effect on the control flow of the executable, which can be represented with a *function call graph*. Research has demonstrated that function call graphs (FCGs) can effectively defeat the polymorphic nature of malware through techniques like graph matching [11, 12, 13, 14, 15] and representation learning [16, 17].

Unfortunately, no large-scale FCG datasets have been made publicly available largely due to the proprietary nature of the data. We introduce MALNET-GRAPH, the largest public graph representation learning database ever constructed, representing a large-scale ontology of software function call graphs. MALNET-GRAPH contains over 1.2 million graphs, averaging over 17k nodes and 39k edges per graph, across a hierarchy of 47 types and 696 families. Compared to the only other public FCG database [18], MALNET-GRAPH offers $927\times$ more graphs, $22\times$ larger graphs on average, and $348\times$ more classes (see Figure 1.4 for a comparison of graph representation learning databases). We provide a detailed analysis of MALNET-GRAPH, discussing its properties and provenance, along with the evaluation of state-of-the-art machine learning and graph neural network techniques. The unprecedented scale and diversity of MALNET-GRAPH offers exciting opportunities to advance the frontiers of cybersecurity through graph representation learning—enabling new discoveries and research into imbalanced classification, explainability and the impact of class hardness.

A Large-Scale Image Database of Malicious Software. (Chapter 6). Computer vision

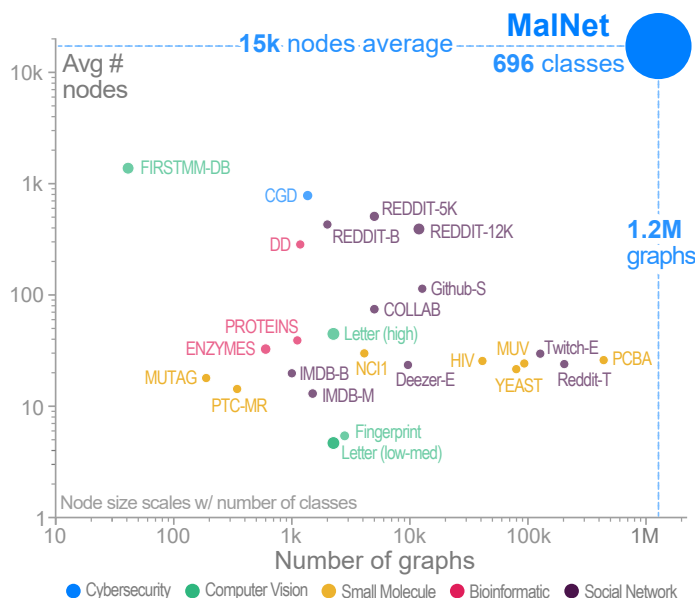


Figure 1.4: MALNET-GRAPH: Advancing State-of-the-Art Graph Databases. MALNET-GRAPH contains 1,262,024 function call graphs averaging 17,242 nodes and 39,043 edges per graph, across a hierarchy of 47 types and 696 families of malware.

is playing an increasingly important role in automated malware detection with the rise of the image-based binary representation. These binary images are fast to generate, require no feature engineering, and are resilient to popular obfuscation methods. Significant research has been conducted in this area, however, it has been restricted to small-scale or private datasets that only a few industry labs and research teams have access to. This lack of availability hinders examination of existing work, development of new research, and dissemination of ideas. We release MALNET-IMAGE, the **largest public cybersecurity image database**, offering **24× more images** and **70× more classes** than existing databases (available at <https://mal-net.org>). MALNET-IMAGE contains over 1.2 million malware images—across 47 types and 696 families—democratizing image-based malware capabilities by enabling researchers and practitioners to evaluate techniques that were previously reported in propriety settings. We report the first million-scale malware detection results on binary images. MALNET-IMAGE unlocks new and unique opportunities to advance the frontiers of machine learning, enabling new research directions into vision-based cyber defenses, multi-class imbalanced classification, and interpretable security.

1.1.4 Part IV: Robust Models

Deep learning has seen rapid development in the fields of cybersecurity and healthcare due to its state-of-the-art performance on a wide range of challenging tasks. However, these

highly advanced architectures are vulnerable to adversarial manipulation, resulting in a variety of system failures. Our research tackles two high-impact societal problems in cybersecurity and healthcare affecting millions of lives—*through the lens of robust deep learning models*—including: (1) the protection of computer vision systems, such as those on self driving cars, by creating robust models that closely align the human visual perception and cognition system with that of machine intelligence (UNMASK); and (2) the development of noise robust deep learning models, which allows us to detect sleep disorders at an earlier stage in the comfort of their own home (REST). Through UNMASK, where we developed deep learning models to identify robust features in image data, we apply this idea to the healthcare setting where we develop models that target robust signal information in EEG data to enable noise robust sleep monitoring.

UnMask: Adversarial Detection and Defense Through Robust Feature Alignment (Chapter 7). To protect computer vision systems from adversarial attack, such as those on self driving cars, we have developed UNMASK, an adversarial detection and defense framework. UNMASK combats adversarial attacks through semantic coherence alignment—extracting robust features (e.g., beak, wings, eyes) from an image (e.g., “bird”) and aligning them with the expected classification features. For example, if the extracted features for a “bird” image are *wheel*, *saddle* and *frame*, the model may be under attack (see Figure 1.5). UNMASK detects such attacks and defends the model by rectifying the misclassification, re-classifying the image based on its robust features. Our extensive evaluation shows that UNMASK *detects* up to 96.75% of attacks, and *defends* the model by correctly classifying up to 93% of adversarial images produced by the current strongest attack, Projected Gradient Descent, in the gray-box setting. UNMASK provides significantly better protection than adversarial training across 8 attack vectors, averaging 31.18% higher accuracy.

REST: Robust and Efficient Neural Networks for Sleep Monitoring in the Wild (Chapter 8). As many as 70 million Americans suffer from sleep disorders that affects their daily functioning, long-term health and longevity. The long-term effects of sleep deprivation and sleep disorders include an increased risk of hypertension, diabetes, obesity, depression, heart attack, and stroke [19]. The cost of undiagnosed sleep apnea alone is estimated to exceed 100 billion in the US [20]. To combat this, significant attention has been devoted towards integrating deep learning technologies in the healthcare domain. However, to safely and practically deploy deep learning models for home health monitoring, two significant challenges must be addressed: the models should be (1) robust against noise; and (2) compact and energy-efficient. We propose REST, a new method that simultaneously tackles both issues via 1) *adversarial training* and controlling the Lipschitz constant of the neural network through *spectral regularization* while 2) enabling neural network

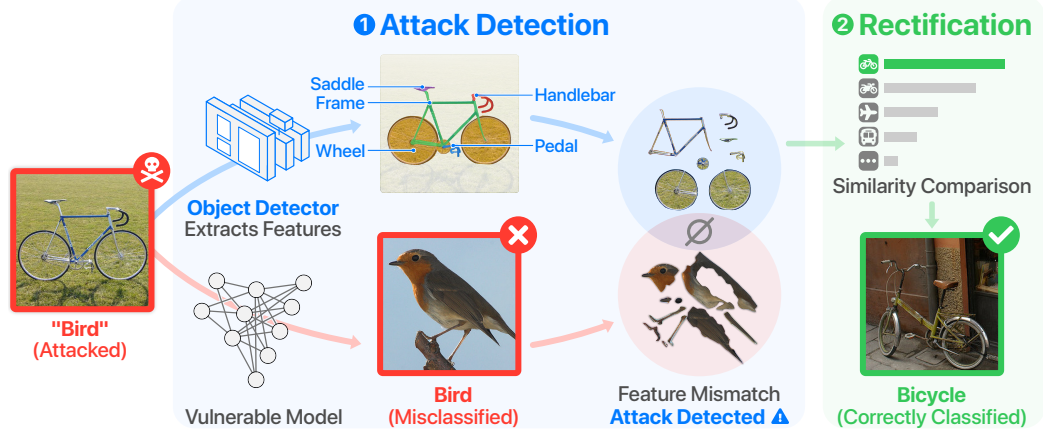


Figure 1.5: UNMASK combats adversarial attacks (in red) through extracting *robust features* from an image (“Bicycle” at top), and comparing them to expected features of the classification (“Bird” at bottom) from the unprotected model. Low feature overlap signals an attack.

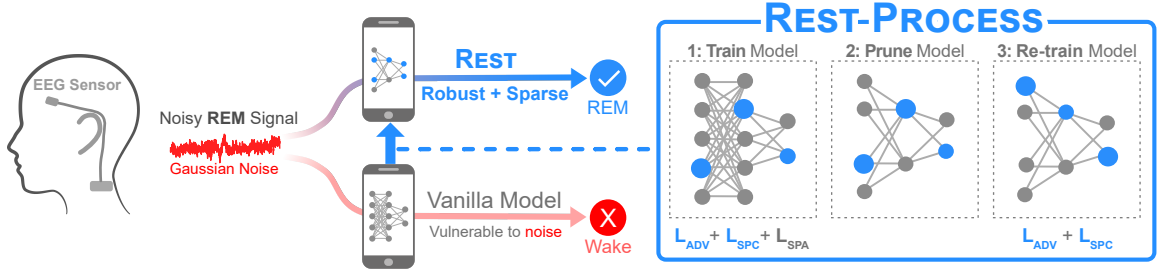


Figure 1.6: REST Overview: (from left) When a noisy EEG signal belonging to the REM (rapid eye movement) sleep stage enters a traditional neural network which is vulnerable to noise, it gets wrongly classified as a Wake sleep stage. On the other hand, the same signal is correctly classified as the REM sleep stage by the REST model which is both robust and sparse. (From right) REST is a three step process involving (1) training the model with adversarial training, spectral regularization and sparsity regularization (2) pruning the model and (3) re-training the compact model.

compression through *sparsity regularization* (see Figure 1.6). We demonstrate that REST produces highly-robust and efficient models that substantially outperform the original full-sized models in the presence of noise. For the sleep staging task over single-channel electroencephalogram (EEG), the REST model achieves a macro-F1 score of 0.67 vs. 0.39 achieved by a state-of-the-art model in the presence of Gaussian noise while obtaining 19× parameter reduction and 15× MFLOPS reduction on two large, real-world EEG datasets. By deploying these models to an Android application on a smartphone, we quantitatively observe that REST allows models to achieve up to 17× energy reduction and 9× faster inference.

1.2 Thesis Statement

To address large-scale societal problems in cybersecurity and healthcare *through the lens of robust machine learning* by developing:

1. *tools* that guide and assist researchers in navigating the complex field of graph robustness;
2. *algorithms* that quantify network vulnerability to lateral attack and guide enterprise system defense resource allocation;
3. *databases* that enable the development of next-generation strong cybersecurity defenses;
4. *models* that are robust to noise and adversarial manipulation, helping guard against surprise attacks and catastrophic failure.

1.3 Research Contributions

New graph algorithms and deep learning models.

- We construct D²M, the first graph theoretic framework that systematically quantifies network vulnerability to lateral attack and identifies at-risk devices (Chapter 4).
- We develop REST, the first noise-robust and efficient deep learning model designed for at-home sleep monitoring by endowing models with noise robustness through (1) *adversarial training* and (2) *spectral regularization*; and promoting energy and computational efficiency by enabling compression through (3) *sparsity regularization* (Chapter 8).
- We contribute UNMASK, the first deep learning framework using semantic coherence to detect and defeat adversarial attacks by quantifying the similarity between the image’s extracted features with the expected features of its predicted class (Chapter 7).

Large-Scale Databases.

- We introduce MALNET-GRAPH, the largest public graph database ever constructed, representing a large-scale ontology of software function call graphs. MALNET-GRAPH contains over 1.2 million graphs, averaging over 17k nodes and 39k edges per graph, across a hierarchy of 47 types and 696 families. Compared to the popular REDDIT-12K database, MALNET-GRAPH offers $105\times$ more graphs, $44\times$ larger graphs on average, and $63\times$ more classes (Chapter 5).

- We introduce MALNET-IMAGE, the largest publicly available cybersecurity image database, offering $24\times$ more images and $70\times$ more classes than existing databases. The scale and diversity of MALNET-IMAGE unlocks new and exciting cybersecurity opportunities to the computer vision community—democratizing image-based malware capabilities by enabling researchers and practitioners to evaluate techniques that were previously reported in propriety settings. (Chapter 6).

Open source tools and knowledge repositories.

- We contribute TIGER, the first open-sourced Python toolbox for graph vulnerability and robustness analysis. TIGER contains 22 graph robustness measures with both original and fast approximate versions; 17 failure and attack strategies; 15 heuristic and optimization based defense techniques; and 4 simulation tools (Chapter 3).
- We distill key findings across numerous graph vulnerability and robustness domains in the form of a survey paper, providing researchers access to crucial knowledge by— (1) summarizing recent and classical graph robustness measures; (2) exploring which robustness measures are most applicable to different domains (e.g., social, infrastructure); (3) reviewing attack strategy effectiveness across network topologies; and (4) extensive discussion on selecting defense techniques to mitigate attacks (Chapter 2).

1.4 Impact

- D²M (Chapter 4) has led to major impact to the **Microsoft Defender Advanced Threat Protection product**, inspiring changes to the product’s approach to detect and prevent lateral movement, well known as one of the most challenging areas of post-breach detection.
- TIGER (Chapter 3) has been integrated into the **Nvidia Data Science Teaching Kit** available to educators across the world; and Georgia Tech’s Data and Visual Analytics class with over 1,000 students.
- UNMASK (Chapter 7) helped win a **multi-million dollar** DARPA GARD (Guaranteeing AI Robustness against Deception) grant.
- MALNET-GRAPH (Chapter 5) represents the **worlds largest graph representation learning database**, enabling new research and discoveries into imbalanced classification, explainability and the impact of class hardness.
- MALNET-IMAGE (Chapter 6) is the **worlds largest binary-image database**, unlocking new and unique opportunities to advance the frontiers of vision-based cyber

defenses, multi-class imbalanced classification, and interpretable security.

- Our innovations in graph mining, deep learning, cybersecurity and healthcare were recognized and invested in by an **IBM PhD Fellowship**, a **Raytheon Fellowship**, and an **NSF Graduate Research Fellowship** (GRFP).

Part I

Robust Tools

Overview

We begin by surveying the field of graph vulnerability and robustness, highlighting its long and diverse history by thoroughly summarizing the state-of-the-art by analyzing over 85 papers and their contributions to the field. Our goal is to guide researchers and practitioners in navigating the expansive field of network robustness, while summarizing answers to key questions. Clicking on the link below will open its PDF version in the browser:

Chapter 2: Graph Vulnerability and Robustness: A Survey. Scott Freitas, Diyi Yang, Srijan Kumar, Hanghang Tong, Duen Horng Chau. [Under Review] *IEEE Transactions on Knowledge and Data Engineering (TKDE)*. Online, 2021. <https://arxiv.org/pdf/2105.00419.pdf>

Through the survey, we find that no comprehensive open-source toolbox currently exists to assist researchers and practitioners in this important topic. This lack of available tools hinders reproducibility and examination of existing work, development of new research, and dissemination of new ideas. To address this, we develop TIGER, the first open-sourced Python toolbox for evaluating network vulnerability and robustness of graphs. TIGER contains 22 graph robustness measures with both original and fast approximate versions when possible; 17 failure and attack mechanisms; 15 heuristic and optimization based defense techniques; and 4 simulation tools. Clicking on the link below will open its PDF version in the browser:

Chapter 3: Evaluating Graph Vulnerability and Robustness using TIGER
Scott Freitas, Diyi Yang, Srijan Kumar, Hanghang Tong, Duen Horng Chau.
ACM International Conference on Information and Knowledge Management (CIKM). Online, 2021. <https://arxiv.org/abs/2006.05648>

CHAPTER 2

GRAPH VULNERABILITY AND ROBUSTNESS: A SURVEY

The study of network robustness is a critical tool in the characterization and sense making of complex interconnected systems such as infrastructure, communication and social networks. While significant research has been conducted in these areas, gaps in the surveying literature still exist. Answers to key questions are currently scattered across multiple scientific fields and numerous papers. In this survey, we distill key findings across numerous domains and provide researchers crucial access to important information by—(1) summarizing and comparing recent and classical graph robustness measures; (2) exploring which robustness measures are most applicable to different categories of networks (e.g., social, infrastructure); (3) reviewing common network attack strategies, and summarizing which attacks are most effective across different network topologies; and (4) extensive discussion on selecting defense techniques to mitigate attacks across a variety of networks. This survey guides researchers and practitioners in navigating the expansive field of network robustness, while summarizing answers to key questions. We conclude by highlighting current research directions and open problems.

2.1 Introduction

There are three fundamental tasks in the study of network robustness: (i) development of measures to quantify network robustness, (ii) identification of network attack mechanisms, and (iii) construction of defensive techniques to resist network failures and recover from attacks. First mentioned as early as the 1970's [21], network robustness has a rich and storied history spanning numerous fields of engineering and science [22, 23, 5, 24]. This diversity of research has generated a variety of unique perspectives, providing fresh insight into challenging problems, while equipping researchers with fundamental knowledge for their investigations. While the fields of study may be diverse, they are linked by a common definition of network robustness [25, 26, 23]:

Robustness is a measure of a network's ability to continue functioning when part of the network is either naturally damaged or targeted for attack.

To provide some intuition for this definition, we consider an example of a power grid network that is susceptible to both *natural* failures and *targeted* attacks. A natural failure happens when a *single* power substation fails due to erosion of parts or natural disasters.

However, when one substation fails, additional load is routed to alternative substations, which can cause *cascading failures*. Not all failures originate from natural causes, some come from *targeted* attacks, such as enemy states hacking into the grid to sabotage key equipment to maximally damage the operations of the electrical grid. Through analyzing and understanding the robustness of these networks, we can mitigate damage from both natural failures and targeted attacks, and in some cases, prevent it altogether.

Unfortunately, the nature of cross-disciplinary research also comes with significant challenges. Oftentimes important discoveries made in one field are not quickly disseminated, leading to missed innovation opportunities. In this survey, our goal is to distill key research questions raised in prior related research [26], that if addressed effectively, will assist readers in understanding the complex interconnections of this topic, and accelerate the dissemination of ideas. Specifically, we analyze and compare numerous classical and modern robustness techniques—addressing a crucial gap in the survey literature, and helping set the stage for future work to be built upon.

2.1.1 Contributions

We distill and summarize critical topics found in the literature of network robustness through contributions C1-C5.

C1. Summary & Comparison of Robustness Measures We summarize 18 modern and classic network robustness measures, along with how each measure is *linked* to the evaluation of graph vulnerability and robustness. We then compare robustness measures through a set of desired robustness measure properties, called axioms, which we identified throughout the literature. Each axiom captures a different property of network robustness, such as the ability to identify alternative pathways (e.g., account for multiple routes between power substations). Our goal is providing researchers a set of tools to objectively compare robustness measures for use in their own applications.

C2. Exploration of Robustness Measure Applications Not all robustness measures are equally applicable to every category of network data. For example, the graph clustering coefficient is important to the study of social networks since it provides an indication of group “tightness”. However, water distribution networks require measures that can account for bottlenecks and alternative pathways. We delve into the literature and summarize why some measures are more applicable to particular domains. In particular, we study two high-impact use case scenarios—(i) transportation networks and (ii) water distribution networks.

C3. Overview of Network Attack Strategies By understanding the topology of the network, we can analyze the effects of natural failures and targeted attacks. We discuss popular

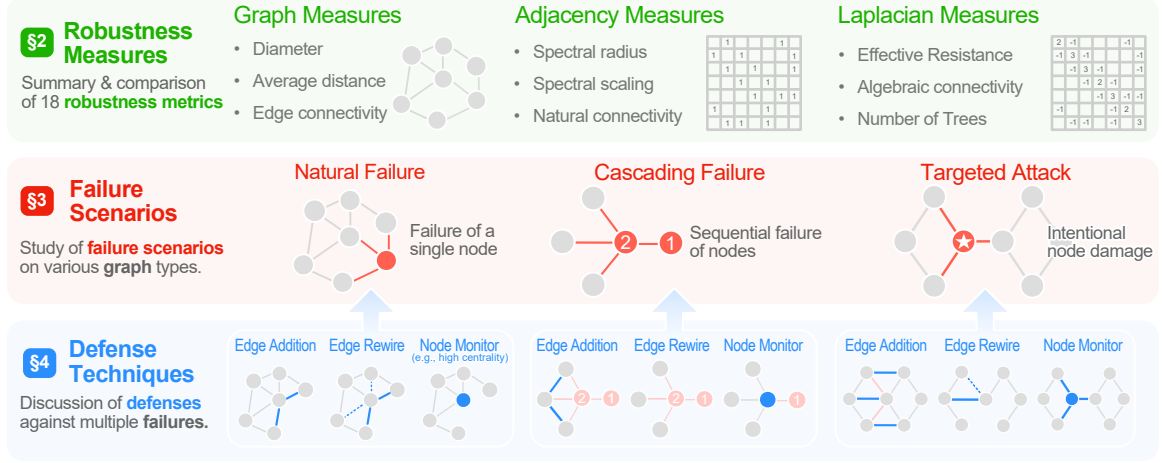


Figure 2.1: A visual overview of the work surveyed . §2 summarizes and compares 18 graph robustness. §3 overviews methods of network failure and attack. §4 summarizes network defense techniques across a variety of graph topologies and attack vectors.

network attack strategies, and provide a high level summary of which attack strategies are best adapted for different network topologies. This attack strategy comparison, as the first of its kind, provides researchers and practitioners a quick guide to effective attack selection.

C4. Comparison of Network Defense Mechanisms By understanding the common network topologies and attack strategies from C3, we can study the effectiveness of network defense mechanisms. We summarize numerous defense mechanisms in relation to various network topologies, mechanisms of natural failure, and targeted attacks across the three primary mechanisms for defending a network:

- (i) *edge rewiring* (e.g., rewire power lines)
- (ii) *edge addition* (e.g., add additional power lines)
- (iii) *node monitoring* (e.g., closely monitor substation)

Each mechanism has an associated benefit and cost, where some are disproportionately more expensive than others. We explore these trade-offs with the goal of providing the reader a comprehensive overview of available defense options across a variety of real world scenarios to assist in the decision making of network defense. To the best of our knowledge, this is the first comprehensive comparison of network defense techniques across attack vectors and network topology.

C5. Highlight Open Problems and Research Directions Through careful analysis of the existing network robustness literature, we identify and distill open problems that have strong potential as future research directions. Promising directions and open problems for

future network robustness research include— (1) an axiomatic study of desired properties in robustness measures, helping guide the selection and development of new measures; (2) interpretability of robustness measures to assist users in understanding the impact of measure scores; (3) applying the study of network robustness to additional high-impact domains such as physical security and cybersecurity; and (4) bridging the study of graph vulnerability and robustness with recent developments in adversarial machine learning on graph structured data.

2.1.2 *Survey Methodology & Summarization Process*

We study an extensive number of existing works and identify three main types of research contributions that they aim to make: (i) study of network robustness measures; (ii) evaluation and development of attacks; and (iii) evaluation and development of defense mechanisms. We frame our work based on these contributions. Doing so allows us to summarize and compare relevant works from computer science, engineering, mathematics, and the sciences. This helps equip researchers with fundamental knowledge for their investigation, and provide them with fresh insights. Specifically, we select works from top journals and conferences from the relevant domains. Table 2.1 lists some of the most prominent publication venues and their acronyms. We also include papers posted on arXiv, an open-access, electronic repository, as many cutting-edge papers are first released here.

As the study of graph robustness has been carried out in a variety of fields (e.g., mathematics, physics, computer science), the terminology often varies from field to field. As such, we refer to the following word pairs interchangeably: (network, graph), (vertex, node), (edge, link), (adversary, attacker). We follow standard notation and use capital bold letters for matrices (e.g., \mathbf{A}), lower-case bold letters for vectors (e.g., \mathbf{a}) and calligraphic font for sets (e.g., \mathcal{S}). Throughout this work we focus our attention on undirected and unweighted graphs, unless otherwise noted. The reader may want to refer to Table 2.3 throughout the survey for technical terms.

2.1.3 *Related Surveys*

While many surveys have been conducted in domain specific applications of network robustness, including water distribution networks [27], airline route networks [28], power grids [29, 30], to our knowledge there is no survey that summarizes the graph robustness landscape at large. Different from all the related articles mentioned above, our survey provides a comprehensive, cross-domain framework to describe graph vulnerability and robustness, discusses the rapidly growing community at large, and presents major research

Table 2.1: Relevant venues in order of journals, conferences, workshops, and preprints. Within each category, we order the venues based on the most recently available impact factors reported officially (e.g., venues’ websites).

Nature	Nature
PR	Physics Reports: A Review Section of Physics Letters
Smart Grid	IEEE Transactions on Smart Grid
PNAS	National Academy of Sciences of the USA
PRL	Physical Review Letters
SIREV	SIAM Review
SMC	IEEE Systems, Man, and Cybernetics: Systems
TKDE	IEEE Transactions on Knowledge & Data Engineering
CNSNS	Communications in Nonlinear Science and Numerical Simulation
KAIS	Knowledge and Information Systems
Physica A	Physica A: Statistical Mechanics and its Applications
RA	Risk Analysis
CHAOS	An Interdisciplinary Journal of Nonlinear Science
PLOS	PLOS ONE
DMKD	Data Mining and Knowledge Discovery
SN	Social Networks: An International Journal of Structural Analysis
PRE	Physical Review E
TOPS	ACM Transactions on Privacy and Security
EPL	A Letters Journal Exploring the Frontiers of Physics
Physica B	Physica B: Condensed Matter
JOCN	Journal of Complex Networks
EPJ B	The European Physical Journal B
CPL	Chinese Physics Letters
FCS	Frontiers of Computer Science
LAA	Linear Algebra and its Applications
Web	The Web Conference (formerly WWW)
KDD	ACM Knowledge Discovery & Data Mining
WSDM	ACM Conference on Web Search & Data Mining
ICDM	IEEE International Conference on Data Mining
CIKM	ACM Information & Knowledge Management
NDSS	Network and Distributed System Security Symposium
INFO	IEEE Conference on Computer Communications
CDC	IEEE Conference on Decision and Control
SDM	SIAM International Conference on Data Mining
PKDD	European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases
PAKDD	Pacific-Asia Conference on Knowledge Discovery & Data Mining
A-POL	Transportation Research Part A: Policy and Practice
GLOBE	IEEE Global Communications Conference
DRCN	Design of Reliable Communication Networks
RNDM	Resilient Networks Design and Modeling Workshop
WORM	ACM Workshop on Rapid Malcode
SIMPLEX	Simplifying Complex Network for Practitioners
arXiv	arXiv.org e-Print Archive

trajectories synthesized from existing literature.

2.1.4 Survey Organization.

Figure 2.1 shows a visual overview of this survey’s structure and Table 2.2 summarizes representative works. The remainder of this chapter is divided into seven parts. Each major component (measures, attacks and defenses) is given one or more sections, ordered to motivate how each component builds on top of the other.

§ 2.2 Summarizing & Comparing Robustness Measures

We summarize recent and classical robustness measures, along with how each measure is *linked* to the evaluation of graph vulnerability and robustness. We then create a table summarizing each robustness measure, allowing users to compare each measures in a simple manner.

§ 2.3 Discussing Network Failures and Targeted Attacks

We discuss natural failures and targeted attacks strategies in relation to common graph topologies.

§ 2.4 Analyzing Network Defense Mechanisms

We summarize common network defense mechanisms that are used to mitigate damage across a variety of network topologies and attacks.

Section 2.5 presents research directions and open problems that we have gathered and distilled from the literature survey. Section 2.6 concludes the survey.

2.2 Robustness Measures

We begin by summarizing 18 recent and classic robustness measures, dividing each measure into one of three categories depending on whether it uses the graph (Section 2.2.1), adjacency matrix (Section 2.2.2) or Laplacian matrix (Section 2.2.3). After describing each measure, we describe its *link* to the study of network robustness. We make note of some additional robustness measures such as scattering number [112], tenacity [113], integrity [114], fault diameter [24], toughness [21] and isoperimetric number [115]. However, we do not consider them for evaluation since they are combinatorial measures for general graphs [47].

Robustness Measures

Attack Defense Where

Work	2.2.1 Binary Connectivity	2.2.1 Vertex Connectivity	2.2.1 Edge Connectivity	2.2.1 Diameter	2.2.1 Average Distance	2.2.1 Avg. Vertex Betweenness	2.2.1 Avg. Edge Betweenness	2.2.1 Global Clustering Coefficient	2.2.1 Largest Connected Component	2.2.2 Spectral Radius	2.2.2 Spectral Gap	2.2.2 Natural Connectivity	2.2.2 Spectral Scaling	2.2.2 Generalized Robustness Index	2.2.3 Algebraic Connectivity	2.2.3 Number of Spanning Trees	2.2.3 Effective Resistance	2.3.3 Node Removal	2.3.3 Edge Removal	2.4.1 Edge Addition	2.4.1 Edge Rewiring	2.4.1 Node Monitoring	Publication Venue
Albert, et.al. [31]																							Nature
Alenazi, et.al. [32]																							RNDM
Alenazi, et.al. [33]																							DRCN
Baig, et.al. [34]																							Web
Baras, et.al. [35]																							CDC
Berdica [36]																							A-POL
Bernstein, et.al. [37]																							INFO
Beygelzimer, et.al. [23]																							Physica A
Bigdeli, et.al. [38]																							SIMPLEX
Bishop, et.al. [39]																							EPL
Bocca, et.al. [40]																							PR
Borgatti, et.al. [41]																							SN
Briesemeis, et.al. [42]																							WORM
Buldyrev, et.al. [43]																							Nature
Byrne, et.al. [44]																							Sandia
Caballero, et.al. [45]																							NDSS
Callaway, et.al. [46]																							PRL
Chan, et.al. [47]																							SDM
Chakrabarti, et.al. [48]																							TOPS
Chan, et.al. [26]																							DMKD
Chen, et.al. [49]																							TKDE
Chen, et.al. [50]																							ICDM
Chen, et.al. [51]																							TKDD
Crucitti, et.al. [52]																							PRE
Dekker [53]																							ACSC
Derrible, et.al. [54]																							Physica A
Duan, et.al. [55]																							Physica A
Ellens, et.al. [56]																							LAA
Ellens, et.al. [25]																							arXiv
Estrada, et.al. [57]																							Physica B
Estrada, et.al. [58]																							EPL
Freitas, et.al. [59]																							SDM
Freitas, et.al. [60]																							arXiv
Gao, et.al. [61]																							PRL
Ghosh, et.al. [62]																							SIREV
Holme, et.al. [63]																							PRE
Holmgren [64]																							RA
Jamakovic, et.al. [65]																							NGI
khalil, et.al. [66]																							KDD
Kinney, et.al. [67]																							EPJ B
Klau, et.al. [68]																							Net. Anal.
Latora, et.al. [69]																							PRE
Le, et.al. [70]																							SDM
Leskovec, et.al. [71]																							KDD
Liu, et.al. [72]																							FCS
Lu, et.al. [73]																							PLOS One
Malliaros, et.al. [74]																							SDM

Table 2.2: Summary of works studied in this survey, each row is one work. Columns are grouped into one of three categories—robustness measures, attacks and defenses—corresponding to primary chapter sections (except “where”). In addition, we divide the robustness measure columns into three categories based on whether it uses the graph, adjacency matrix, or Laplacian matrix, from left to right, respectively (using dashed lines)

Work	2.2.1	2.2.1	2.2.1	2.2.1	2.2.1	2.2.1	2.2.1	2.2.1	2.2.1	2.2.2	2.2.2	2.2.2	2.2.2	2.2.2	2.2.3	2.2.3	2.2.3	2.3.3	2.3.3	2.4.1	2.4.1	2.4.1	Venue
Marzo,et.al.[75]																							RNDM
Mattsson,et.al.[76]																							A-POL
Mieghem,et.al.[77]																							PRE
Milanese,et.al.[78]																							PRE
Motter,et.al.[79]																							PRE
Nardo,et.al.[80]																							Water
Nguyen,et.al.[81]																							Smart Grid
Parandeh,et.al.[82]																							GLOBE
Parshani,et.al.[83]																							PRL
Paul,et.al.[84]																							EPJ B
Prakash,et.al.[85]																							PKDD
Prakash,et.al.[86]																							KAIS
Prakash,et.al.[87]																							SDM
Rueda,et.al.[88]																							JNSM
Saha,et.al.[89]																							SDM
Schneider,et.al.[90]																							PNAS
Schneider,et.al.[91]																							PRE
Shao,et.al.[92]																							PRE
Shargel,et.al.[93]																							PRL
Sydney,et.al.[94]																							arXiv
Tanaka,et.al.[95]																							Nature
Tong,et.al.[5]																							ICDM
Tong,et.al.[96]																							CIKM
Torres,et.al.[97]																							arXiv
Trajanovski,et.al.[98]																							JOCN
Vespignani,et.al.[99]																							Nature
Wang,et.al.[100]																							Physica A
Wang,et.al.[101]																							EPJ B
Watts,et.al.[102]																							Nature
Wu,et.al.[103]																							CPL
Wu,et.al.[104]																							SMC
Xia,et.al.[105]																							Physica A
Yang,et.al.[106]																							PLOS ONE
Yazdani,et.al.[107]																							CNSNS
Yazdani,et.al.[108]																							CHAOS
Zeng,et.al.[109]																							PRE
Zhao,et.al.[110]																							PRE
Zhao,et.al.[111]																							PLOS ONE

2.2.1 Measures Based on Graph Connectivity

We review 9 of the most common graph robustness measures, each which takes as input an undirected, unweighted graph $G = (\mathcal{V}, \mathcal{E})$, where \mathcal{V} is the set of vertices, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges. We let $n = |\mathcal{V}|$ and $m = |\mathcal{E}|$ as the number of vertices and number of edges, respectively.

Table 2.3: Symbols and Definition Tables. We divide symbols and definitions based on whether it corresponds to use with the graph, adjacency matrix or Laplacian matrix. From left to right, symbol and definition tables for the graph, adjacency matrix and Laplacian matrix.

Symbol	Graph Definitions
$G(\mathcal{V}, \mathcal{E})$	graph G , set of nodes \mathcal{V} , edges \mathcal{E}
n, m	number nodes $ V $, edges $ \mathcal{E} $
$\kappa, \kappa_v, \kappa_e$	binary, vertex, edge connectivity
\bar{d}	average geodesic distance
d_{max}	graph diameter
\bar{b}_v, \bar{b}_e	avg. vertex, edge betweenness
C	global clustering coefficient
L	largest connected component

Symbol	Adjacency Matrix Definitions
\mathbf{A}	adjacency matrix
$\mathbf{A}_{i,j}$	element at i th row, j th col.
$\mathbf{u}(i)$	eigenvector at position i
$\rho = \lambda_1$	spectral radius = 1st eigenvalue
λ_d	spectral gap
$\bar{\lambda}$	natural connectivity
ξ	spectral scaling
r_k	generalized robustness index

Symbol	Laplacian Matrix Definitions
\mathbf{L}	laplacian matrix
$\mathbf{L}_{i,j}$	element at i th row, j th col.
\mathbf{D}	diagonal matrix
d_i	degree of node i
μ_1	smallest eigenvalue of \mathbf{L}
μ_2	algebraic connectivity of \mathbf{L}
T	number of spanning trees
R	effective resistance

Binary Connectivity (κ)

A classical graph measure which determines whether or not a graph is connected ($\kappa=1$) or unconnected ($\kappa=0$) by examining whether all pairs of vertices have a connecting path. A graph is considered unconnected if at least one pair of vertices does not have a connecting path. This can be calculated using breadth-first or depth-first search, starting at any vertex with time complexity $O(n + m)$.

Robustness link. Practically speaking, binary connectivity is a poor measure of network robustness since it only identifies whether a network is disconnected.

Vertex Connectivity (κ_v)

An extension of binary connectivity, vertex connectivity $\kappa_v(G)$ is the minimal number of vertices that need to be removed to disconnect the graph $\kappa(G) = \min\{\kappa_v(u, v) \mid \text{unordered pair } (u, v) \in G\}$. Assume $(u, v) \notin E$, then $\kappa_v(u, v)$ is the minimal number of vertices that would destroy every path between vertices u and v . If $(u, v) \in E$, then $\kappa_v(u, v) = n - 1$ [116]. Computing $\kappa(G)$ can be reduced to a max-flow problem with a time complexity of $O(n^{8/3}m)$ [117].

Robustness link. This measure has a natural relationship to the robustness of the graph, since $\kappa_v(G)$ increases as the graph becomes harder to disconnect.

Edge Connectivity (κ_e)

Also an extension of binary connectivity, edge connectivity $\kappa_e(G)$ is the minimal number of edges that can be removed to disconnect the graph $\kappa_e(G) = \min\{\kappa_e(u, v) \mid \text{unordered pair } (u, v) \in G\}$. Let u, v , be a pair of distinct vertices in graph G , $\kappa_e(u, v)$ is the minimal number of deleted edges that would disconnect all paths between u and v . Calculating $\kappa_e(u, v)$ for each unordered pair, we can ascertain the minimal edge connectivity. The best known algorithm has a time complexity of $O(nm)$ [117, 116]. For an incomplete graph, an interesting property of vertex and edge connectivity is that $\kappa_v(G) \leq \kappa_e(G) \leq \delta_{\min}(G)$, where $\delta_{\min}(G) = \min\{\deg(v) \mid v \in G\}$ [118].

Robustness link. Similar to κ_v , this measure ties to the naturally ties to graph robustness since $\kappa_e(G)$ increases as the graph becomes harder to disconnect.

Diameter (d)

The diameter d of a connected graph is the longest shortest path between all pairs of nodes $d(G) = \max\{d(u, v) \mid \text{unordered pair } (u, v) \in G\}$, where $d(u, v)$ is the shortest path between vertices u and v . In order to calculate the diameter of a network all pairs of shortest paths need to be computed, requiring a time of complexity of $O(n^3)$ using the Floyd–Warshall algorithm [119].

Robustness link. The diameter has an intuitive connection to robustness, where a decreasing diameter implies better robustness due to increased connectivity.

Average Distance (\bar{d})

The average geodesic distance \bar{d} in Equation 2.1 provides a measure of network connectivity by calculating the average distance between all pairs of nodes in the graph.

$$\bar{d} = \frac{2}{n(n-1)} \sum_{u \in V} \sum_{\substack{v \in V \\ u \neq v}} d(u, v) \quad (2.1)$$

For each unordered pair of nodes $(u, v) \in G$ in the graph, the shortest path $d(u, v)$ is computed using the Floyd-Warshall algorithm [119], then summed and normalized by $\frac{2}{n(n-1)}$ to account for bi-directional paths $d(u, v) = d(v, u)$ in undirected graphs. This measure is commonly modified to account for disconnected graphs by computing the average inverse distance, also known as the *efficiency*, as shown in Equation 2.2. Unfortunately, both average distance and efficiency have a time complexity of $O(n^3)$ due to the all pairs shortest path computation.

$$\bar{d} = \frac{2}{n(n-1)} \sum_{u \in V} \sum_{\substack{v \in V \\ u \neq v}} \frac{1}{d(u, v)} \quad (2.2)$$

Robustness link. Average distance has a close relation to network connectivity, where a small average distance implies a more robust graph. In contrast, a larger efficiency implies a more robust graph due to inverse computation.

Average Vertex Betweenness (\bar{b}_v)

The average vertex betweenness \bar{b}_v in Equation 2.3 is the summation of vertex betweenness b_u for every node $u \in V$,

$$\bar{b}_v = \sum_{u \in V} b_u \quad (2.3)$$

where vertex betweenness b_u for node u is defined in Equation 2.4 as the number of shortest paths that pass through u out of the total possible shortest paths.

$$b_u = \sum_{s \in V} \sum_{\substack{t \in V \\ s \neq t \neq u}} \frac{n_{s,t}(u)}{n_{s,t}} \quad (2.4)$$

Here $n_{s,t}(u)$ is the number of shortest paths between s and t that pass through u and $n_{s,t}$ is the total number of shortest paths between s and t [120]. Interestingly, the average vertex betweenness \bar{b}_v can be represented as a linear function of the average distance \bar{d} [25], as

shown in Equation 2.5, implying a strong connection between the robustness properties of average vertex betweenness and average distance.

$$\bar{b}_v = \frac{1}{2}(n-1)(\bar{d}+1) \quad (2.5)$$

Computing the vertex betweenness centrality b_u for a single node u has a time complexity of $O(nm)$ using Brandes' algorithm [121]. Therefore, the average vertex betweenness \bar{b}_v has a time complexity of $O(n^2m)$.

Robustness link. Average vertex betweenness has a natural connection to graph robustness since it measures average network throughput of the vertices. The smaller the average the more robust the network, since load is more evenly distributed across each node.

Average Edge Betweenness (\bar{b}_e)

Similar to vertex betweenness, edge betweenness is defined as the number of shortest paths that pass through an edge e out of the total possible shortest paths. Since the formula and intuition for calculating average edge betweenness is nearly identical to average node betweenness, we define it Equation 2.2.1 below and refer the reader to Section 2.2.1 for additional detail.

$$\bar{b}_e = \sum_{e \in E} \sum_{s \in V} \sum_{\substack{t \in V \\ s \neq t}} \frac{n_{s,t}(e)}{n_{s,t}} \quad (2.6)$$

Similar to \bar{b}_v , average edge betweenness can be represented as a linear function of the average distance \bar{d} ,

$$\bar{b}_e = \frac{n(n-1)}{2m} \bar{d} \quad (2.7)$$

implying that some of the robustness properties for \bar{d} will hold for \bar{b}_e as well. The time complexity for average edge betweenness is also $O(n^2m)$.

Robustness link. Average edge betweenness has similar robustness properties to average vertex betweenness. The smaller the average the more robust the network, since load is more evenly distributed across each edge.

Global Clustering Coefficient (C)

The global clustering coefficient C is based on the number of triplets of nodes in the graph, and provides an indication of how well nodes tend to cluster together. By definition, a triplet

is three nodes connected by either two edges (open triplet) or three edges (closed triplet); where a closed triplet is called a triangle. Each triangle contains three closed triplets, one centered on each node. In order to measure the global clustering coefficient, we count the number of closed triplets (or 3x the number of triangles) and divide it by the total number of both closed and open triplets, as in Equation 2.8.

$$C = \frac{\text{closed triplets}}{\text{closed triplets} + \text{open triplets}} \quad (2.8)$$

Alternatively, we can view the global clustering coefficient as the average possible fraction of interconnections among each node $v \in G$ as in Equation 2.9

$$C = \frac{1}{n} \sum_{v \in V} \frac{2 \cdot N_v}{\delta_v(\delta_v - 1)} \quad (2.9)$$

where N_v is the number of edges between neighbors of v , and δ_v is the degree of node v . The time complexity for computing the global clustering coefficient is $O(n \cdot d_{max}^2)$, where d_{max}^2 is the size of the largest adjacency list across all vertices in the graph [122].

Robustness link. A larger global clustering coefficient implies a more robust network since the number of triangles in the graph corresponds to the number of available communication paths.

Largest Connected Component (L)

This measure provides an indication of a graph's connectivity by measuring the fraction of nodes contained in the largest connected component. This is calculated by determining the maximal set of vertices $S \subset V$ such that for each $u \in S$ and $v \in S$, there exists a path from u to v in G . Intuitively, L provides a measure of network availability i.e., what percentage of the nodes can be reached, and is measured as shown in Equation 2.10.

$$L = \frac{|\{d(u, v) \neq \infty \mid \text{unordered pair}(u, v) \in G\}|}{n} \quad (2.10)$$

The time complexity to find the largest connected component is $O(n + m)$, since each BFS (or DFS) call takes linear time in the number of edges and vertices for each component; and since each component is only searched once, all searches are linear in the number of edges and vertices.

Robustness link. This is useful as a measurement of robustness since as the number of removed nodes and edges increases, there reaches a critical fraction where the network connectivity begins to collapse; which can be measured through the size of the largest

connected component.

2.2.2 Measures Based on Adjacency Matrix Spectrum

The adjacency matrix \mathbf{A} is a common network representation, often used when enumerate walks [123]. Formally, we say that $\mathbf{A}(G)$ is the adjacency matrix of G , where $\mathbf{A} \in \{0, 1\}^{n \times n}$ and $\mathbf{A}_{i,j} = \mathbf{A}_{j,i} = 1$ if vertex v_i and v_j are adjacent and $\mathbf{A}_{i,j} = \mathbf{A}_{j,i} = 0$ otherwise. This can be seen in Equation 2.11.

$$\mathbf{A}_{ij} = \begin{cases} 1, & \text{if } i \text{ is adjacency to } j \\ 0, & \text{otherwise} \end{cases} \quad (2.11)$$

It follows that $\mathbf{A}(G)$ is a real symmetric matrix with real eigenvalues $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n$. The set of eigenvalues $\{\lambda_1, \lambda_2, \dots, \lambda_n\}$ is called the spectrum of \mathbf{A} , with corresponding eigenvectors $\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_n$. Several robustness measures have been based on the spectrum of the adjacency matrix; we address five of the most common ones below.

Spectral Radius (ρ)

The largest eigenvalue λ_1 of an adjacency matrix \mathbf{A} is called the spectral radius ρ . The spectral radius is closely related to the *path capacity* or *loop capacity* of the graph. That is, the number of walks of length k ($k = 2, 3, 4, \dots$) gives an indication of how well connected the graph is. If the graph has many loops and paths, then the graph is well connected i.e., larger λ_1 [5, 123, 77]. It has been shown in [124] that the spectral radius is also closely tied to the epidemic threshold τ of a network in the flu-like SIS (susceptible-infected-susceptible) model. In particular, they prove that $\frac{\beta}{\delta} < \tau = \frac{1}{\lambda_1}$, where β is the birth rate of a virus or disease and δ is the cure rate. This means for a given virus strength, an epidemic is more likely to occur on a graph with larger λ_1 . While this seems contradictory at first glance, increased vulnerability to virus propagation actually implies a graph is more robust to natural failures and targeted attacks. The time complexity for computing the spectral radius is $O(m)$, since computing the first eigenpair in sparse matrix form is linear with respect to the number of edges [49].

Robustness link. As a robustness measure, a larger λ_1 indicates a more robust graph to random failures and attack, along with increased susceptibility to virus propagation [70, 5, 49, 89, 96, 26]. This is because the backup paths and redundancy in a network are the same mechanisms that allow a virus to quickly propagate.

Spectral Gap (λ_d)

The difference between the largest and second largest eigenvalues of the adjacency matrix ($\lambda_1 - \lambda_2$) is called the spectral gap λ_d . As a robustness measure, the spectral gap is a simple way to estimate the robustness of a graph—if the spectral gap is large, the graph shows good robustness; if it is small, the robustness is poor [26, 74]. This is because a large spectral gap is indicative of a network with good expansibility properties, while a small spectral gap indicates a network with bottlenecks and bridges [57]. This ability to communicate the existence of bridges in the graph is a unique property of the spectral gap, and not found in the spectral radius. The time complexity to compute the spectral gap is $O(m + n)$ [125].

Robustness link. A unique link between the spectral gap and network robustness is the ability to identify bridges and bottlenecks in the graph [57]. Unfortunately, it is not evident how large the spectral gap needs to be for a graph to be characterized as robust [74].

Natural Connectivity ($\bar{\lambda}$)

Natural connectivity can be interpreted as the “average eigenvalue” of the adjacency matrix $\bar{\lambda}$ [103], and is defined in Equation 2.12.

$$\bar{\lambda}(G) = \ln\left(\frac{1}{n} \sum_{i=1}^n e^{\lambda_i}\right) \quad (2.12)$$

Natural connectivity has a physical and structural interpretation that is tied to the connectivity properties of a network. It is often used to measure the availability of alternative pathways in a network, through the weighted number of closed walks [47]. A walk of length k in a graph is defined as an alternating sequence of vertices and edges $v_0 e_1 v_1 e_2 \dots e_k v_k$, where $v_i \in V$ and $e_i = (v_{i-1}, v_i) \in E$. A walk is said to be closed if $v_0 = v_k$. Closed walks are closely related to the *natural connectivity* and *subgraph centrality* (SC), which we can relate through Equation 2.13.

$$SC(G) = \sum_{i=1}^n SC(i) = \sum_{i=1}^n \sum_{k=0}^{\infty} \frac{(A^k)_{ii}}{k!} = \sum_{i=1}^n \sum_{k=0}^{\infty} \frac{\lambda_i^k}{k!} = \sum_{i=1}^n e^{\lambda_i} \quad (2.13)$$

Here $(A^k)_{ii}$ is the number of closed walks of length k on node i , where $k!$ scales the weighted sum so that it does not diverge and longer walks are counted less [126, 103, 47]. We further explore the relationship between subgraph centrality and robustness in Sections 2.2.2 and 2.2.2. For now, we rewrite Equation 2.13 in relation to natural connectivity

as shown in Equation 2.14.

$$\bar{\lambda}(G) = \ln\left(\frac{1}{n} \sum_{i=1}^n e^{\lambda_i}\right) = \ln\left(\frac{1}{n} SC(G)\right) \quad (2.14)$$

Natural connectivity has a time complexity of $O(n^3)$ since it computes the full spectrum of the adjacency matrix [80].

Robustness link. Natural connectivity has a close relation to network topology and dynamical processes on the graph [47]. This can be seen by its close connection to the number of closed walks, which naturally captures the notion of network connectivity and alternative pathways in a network. A larger $\bar{\lambda}$ indicates a more robust graph.

Spectral Scaling (ξ)

When a network is both sparse and highly connected it is said to have “good expansion” (GE), also known as an expander graph or a “good expansion network” (GEN) [127, 57]. Intuitively, we can think of an expander graph as a network lacking bridges or bottlenecks, making it hard to disconnect through the removal of a few nodes or edges. GE is closely related to the spectral gap λ_d of a network, and can be used to identify the GE property if λ_d is sufficiently large (i.e., $\lambda_1 \gg \lambda_2$) [57]. The question is, how large should the spectral gap be in order for a network to be considered a GEN?

Estrada [57] proposes to solve this problem by combining the *spectral gap* with *sub-graph centrality* (SC). In particular, they propose to use odd subgraph centrality SC_{odd} to measure the number of odd length closed walks that a node participates in. This helps to avoid trivial closed walks (i.e., paths) occurring from even length closed walks SC_{even} . Rewriting Equation 2.13, we can view SC in terms of its even and odd components [128]:

$$\begin{aligned} SC(i) &= \sum_{j=1}^n \mathbf{u}_j(i)^2 \cosh(\lambda_j) + \sum_{j=1}^n \mathbf{u}_j(i)^2 \sinh(\lambda_j) \\ &= SC_{even}(i) + SC_{odd}(i) \end{aligned} \quad (2.15)$$

Expanding SC_{odd} into two components based on the first and subsequent eigenvalues:

$$SC_{odd}(i) = [\mathbf{u}_1(i)]^2 \sinh(\lambda_1) + \sum_{j=2} [\mathbf{u}_j(i)]^2 \sinh(\lambda_j) \quad (2.16)$$

Since we know that networks with good expansibility have $\lambda_1 \gg \lambda_2$, we can say that:

$$[\mathbf{u}(i)]^2 \sinh(\lambda_1) \gg \sum_{j=2} [\mathbf{u}_j(i)]^2 \sinh(\lambda_j) \quad (2.17)$$

and therefore rewrite Equation 2.16 using the inequality from Equation 2.17 as follows:

$$SC_{odd}(i) \approx [\mathbf{u}_1(i)]^2 \sinh(\lambda_1) \quad (2.18)$$

From Equation 2.18 we observe that the subgraph centrality is proportional to the first eigenvector \mathbf{u}_1 , yielding the following equation:

$$\mathbf{u}_1(i) \propto \mathbf{A}[SC_{odd}(i)]^\eta \quad (2.19)$$

where $\mathbf{A} = [\sinh(\lambda_1)]^{-0.5}$ and $\eta \approx 0.5$. This implies a linear correlation between $\mathbf{u}_1(i)$ and $SC_{odd}(i)$ for networks with good expansion. In a log-log scale Equation 2.19 can be rewritten as:

$$\log[\mathbf{u}_1(i)] = \log \mathbf{A} + \eta \log[SC_{odd}(i)] \quad (2.20)$$

As a result, a log-log plot of $\mathbf{u}_1(i)$ vs. $SC_{odd}(i)$ shows a linear fit with slope $\eta \approx 0.5$ with an intercept of $\log[\mathbf{A}]$ if the network has GE. In order to quantify this into a robustness measure, [57] proposes the formula for *spectral scaling* in Equation 2.21.

$$\xi(G) = \sqrt{\frac{1}{n} \sum_{i=1}^n \{\log[\mathbf{u}_1(i)] - [\log \mathbf{A} + \eta \log[SC_{odd}(i)]]\}^2} \quad (2.21)$$

We note that spectral scaling calculates $SC_{odd}(i)$ using the full spectrum of the adjacency matrix, not just the first eigenpair. As such, the time complexity is $O(n^3)$ due to the computation of the full adjacency matrix spectrum [129].

Robustness link. As a robustness measure, the closer the value of ξ to zero, the better the expansion properties and the more robust the network. Formally, a network is considered to have GE if $\xi < 10^{-2}$, the correlation coefficient $r < 0.999$ and the slope is 0.5. While this method improves upon the spectral gap, it still suffers from a few shortcomings, including—(i) not scalable to large graphs due to computing the full adjacency matrix spectrum; and (ii) not applicable to bipartite graphs which do not contain odd length closed walks.

Generalized Robustness Index (r_k)

This measure is a fast approximation of *spectral scaling*, which includes only the top k eigenpairs in the subgraph centrality calculation, since only the first few eigenvalues are large, with most of the eigenvalues symmetric around zero. In many real world graphs $k \ll n$, and therefore, k can be considered a low-rank approximation of the adjacency matrix \mathbf{A} [74]. This allows us to compute only a few eigenpairs of \mathbf{A} while capturing a majority of the spectrum information; making the measure scalable to large graphs. This process can be seen in Equation 2.22, where the modified subgraph centrality is referred to as *normalized subgraph centrality* (NSC).

$$NSC_k(i) = \sum_{j=1}^k \sinh(\lambda_j), \forall i \in \mathcal{V} \quad (2.22)$$

Based on the normalized subgraph centrality NSC_k , [74] proposes the *generalized robustness index* r_k :

$$r_k = \sqrt{\frac{1}{n} \sum_{i=1}^n \{ \log[\mathbf{u}_1(i)] - [\log \mathbf{A} + \frac{1}{2} \log[NSC_k(i)]] \}^2} \quad (2.23)$$

where $\mathbf{A} = [\sinh(\lambda_1)]^{-0.5}$. In practice, k can be extremely small while still achieving high accuracy ($k \leq 30$) [74]. This reduces the time complexity of spectral scaling to either $O(kn^2)$ [129] or $(mk + nk^2)$ [125] depending on the implementation.

Robustness link. As a robustness measure, a smaller r_k implies better robustness. Besides from the computational benefit of the low rank approximation, this measure is very similar to spectral scaling. We refer the reader to Section 2.2.2 for additional information.

2.2.3 Measures Based on Laplacian Matrix Spectrum

The Laplacian matrix \mathbf{L} is often used when a problem can be related to spanning trees or the incidence of vertices and edges [123]. Formally, we say that $\mathbf{L}(G)$ is the Laplacian matrix of G , where $\mathbf{L} = \mathbf{D} - \mathbf{A}$ and \mathbf{D} is the diagonal matrix with the degree on the diagonals. It follows that $\mathbf{L}_{i,j} = d_i$ if $i = j$; $\mathbf{L}_{i,j} = -1$ if i is adjacent to j ; and $\mathbf{L}_{i,j} = 0$ otherwise; where d_i is the degree of node i . This can be seen in Equation 2.24.

$$\mathbf{L}_{ij} = \begin{cases} d_i, & \text{if } i = j \\ -1, & \text{if } i \text{ is adjacency to } j \\ 0, & \text{otherwise} \end{cases} \quad (2.24)$$

Since D and A are both symmetric and have real eigenvalues and an orthonormal basis of eigenvectors, the Laplacian matrix is positive semi-definite with nonnegative eigenvalues; with the smallest eigenvalue always being 0. Hence we order the eigenvalues as follows $0 = \mu_1 \leq \mu_2 \leq \dots \leq \mu_n$, where the set of eigenvalues $\{\mu_1, \mu_2, \dots, \mu_n\}$ is called the spectrum of L . Several robustness measures have been based on the spectra of the Laplacian matrix; we address 3 of the most prominent ones below.

Algebraic Connectivity (μ_2)

The second smallest eigenvalue of the Laplacian is called the algebraic connectivity μ_2 , also known as the Fiedler vector [130]. Since the Laplacian is symmetric, positive semi-definite and the rows sum up to 0, the eigenvalues are real and non-negative, with the smallest eigenvalue being zero; and the multiplicity of the zero eigenvalue related to the number of connected components. For a disconnected graph, this means the algebraic connectivity is always zero. The time complexity to compute the algebraic connectivity is $O(m+n)$ [131]

Robustness link. The larger the algebraic connectivity μ_2 the more robust the graph. This can be understood from its relationship to the characteristic path length of a network; and from its connection to node connectivity κ_v and edge connectivity κ_e of a graph, where μ_2 serves as a lower bound $0 \leq \mu_2 \leq \kappa_v \leq \kappa_e \leq d_{min}$. This means that a network with larger algebraic connectivity is harder to disconnect (i.e., more edges, nodes need to be removed) [26].

Number of Spanning Trees (T)

A spanning tree is a subgraph of G containing n nodes, $n-1$ edges and no cycles. We can visualize this as a graph connecting all the vertices with the minimum possible number of edges. The number of spanning trees T is the number of unique spanning trees that can be found in a graph. This measure was originally suggested as an indicator of a graph's ability to stay connected [132]; where Baras and Hovareshti expanded upon it as an indicator of network robustness [35]. As a consequence of the Kirchoff's matrix-tree theorem [133], the number of spanning trees T can be written as a function of the Laplacian eigenvalues as shown in Equation 2.25.

$$T = \frac{1}{n} \prod_{i=2}^n \mu_i \quad (2.25)$$

The time complexity for this measure is $O(n^3)$ due to the computation of the full Laplacian spectrum [134].

Robustness link. The larger the number of spanning trees the more robust the graph. This can be viewed from the perspective of network connectivity, where a larger set of spanning trees provides a measure of alternative pathways. Unfortunately this measure does not work for disconnected graphs since a spanning tree must include all vertices by definition.

Effective Resistance (R)

This measure views a graph as an electrical circuit where an edge (i, j) corresponds to a resistor of $r_{ij} = 1$ Ohm and a node i corresponds to a junction. As such, the effective resistance between two vertices i and j , denoted R_{ij} , is the electrical resistance measured across nodes i and j when calculated using Kirchoff's circuit laws. Extending this measure to the whole graph, we say the *effective graph resistance* R is the sum of resistances for all distinct pairs of vertices [25, 62]. Klein and Randic proved this can be calculated based on the sum of the inverse non-zero Laplacian eigenvalues [22] as shown in Equation 2.26

$$R = \frac{1}{2} \sum_{i,j} R_{ij} = n \sum_{i=2}^n \frac{1}{\mu_i} \quad (2.26)$$

In addition, it has been shown that the effective resistance can be bounded by the algebraic connectivity [56] as shown in Equation 2.27

$$\frac{n}{\mu_2} < R \leq \frac{n(n-1)}{\mu_2} \quad (2.27)$$

The effective resistance has time complexity $O(n^3)$ due to computation of the full Laplacian spectrum.

Robustness link. As a robustness measure, effective resistance measures how well connected a network is, where a smaller value indicates a more robust network [62, 25]. In addition, the effective resistance has many desirable properties, including the fact that it strictly decreases when adding edges [56], and takes into account both the number of paths between node pairs and their length.

2.2.4 Comparing Robustness Measures

In Table 2.4, we highlight each robustness measure, the category it belongs to (graph, adjacency, Laplacian), and its application to network robustness. By distilling all of this measure information into a single table, users can easily compare robustness measures. This greatly assists in selecting robustness measures across domain specific criteria, where the user may have an idea of what the robustness measure needs to incorporate. For example,

Table 2.4: Comparison of robustness measures. Measures are grouped based on whether they use the *graph*, *adjacency* or *Laplacian* matrix. For each measure, we briefly describe it’s application to measuring network robustness.

Robustness Measure	Category	Application to Network Robustness			
Vertex connectivity	graph	higher value	⇒	harder to disconnect graph	⇒ higher robustness
Edge connectivity	graph	higher value	⇒	harder to disconnect graph	⇒ higher robustness
Diameter	graph	lower value	⇒	stronger connectivity	⇒ higher robustness
Average distance	graph	lower value	⇒	stronger connectivity	⇒ higher robustness
Average inverse distance	graph	higher value	⇒	stronger connectivity	⇒ higher robustness
Average vertex betweenness	graph	lower value	⇒	more evenly distributed load	⇒ higher robustness
Average edge betweenness	graph	lower value	⇒	more evenly distributed load	⇒ higher robustness
Global clustering coefficient	graph	higher value	⇒	more triangles	⇒ higher robustness
Largest connected component	graph	higher value	⇒	better connected graph	⇒ higher robustness
Spectral radius	adjacency	larger value	⇒	stronger connectivity	⇒ higher robustness
Spectral gap	adjacency	higher value	⇒	fewer bottlenecks	⇒ higher robustness
Natural connectivity	adjacency	higher value	⇒	more alternative pathways	⇒ higher robustness
Spectral scaling	adjacency	lower value	⇒	fewer bottlenecks	⇒ higher robustness
Generalized robustness index	adjacency	lower value	⇒	fewer bottlenecks	⇒ higher robustness
Algebraic connectivity	laplacian	higher value	⇒	harder to disconnect	⇒ higher robustness
Number of spanning trees	laplacian	higher value	⇒	more alternative pathways	⇒ higher robustness
Effective resistance	laplacian	lower value	⇒	more alternative pathways	⇒ higher robustness

users working with critical infrastructure systems wants to select a robustness measure that takes into account backup pathways, this information is now readily available, along with how to interpret the measure (e.g., lower is better).

2.2.5 Selecting a Robustness Measure

In this section we explore which robustness metrics may be well suited to particular types of network data through an analysis of two high impact use case scenarios—(i) transportation networks and (2) water distribution networks. We begin by exploring the domain specific robustness problems in each use case scenario, and identify important network properties. Studying these network properties in conjunction with the current methods of robustness analysis for the domain, we highlight alternative robustness measures (based on Table 2.4) that have potential to improve performance for the task at hand. Through this process we take a first step in addressing how to select a robustness measure by providing a template for future domain specific network robustness analysis.

Scenario 1: Transportation Networks. Societal dependence on transportation networks (e.g., roadway) is enormous, and only increasing in number and complexity. Unfortunately,

these networks are often purposefully designed to operate with minimal redundancy and high capacity in order to minimize costs. As a result, they are extremely sensitive to failures and disruptions [76]. As such, understanding and studying the vulnerability and robustness of these networks is critical.

While there is currently no definitive definition of transport system vulnerability [76], a well received and representative one is suggested by Berdica [36]: “*Vulnerability in the road transportation system is a susceptibility to incidents that can result in considerable reductions in road network serviceability*”. In order to understand the nature and extent of the vulnerability posed to transportation networks, graph theoretic measures have evolved as a natural tool of representation. In the study of transportation systems it has been suggested robustness measures take into account factors like:

- (i) *average distance* between different stops [76]
- (ii) *alternative pathways* of transport [54]
- (iii) *bottlenecks* inside and between communities [55]

Currently, highly interpretable network analysis tools such as average distance, betweenness centrality, clustering coefficient and largest connected component are proposed as measures of system robustness [55, 76]. From a decision making perspective, these measures are highly interpretable helping inform decision making policy.

Potential Alternatives. Traditional robustness analysis applied to transportation networks could benefit from the use of spectral based techniques, which are scalable to large networks and could potentially provide better performance. In this domain we identified alternative pathways (i.e. redundancy) and bottlenecks as important criteria in robustness measure selection. Therefore the *spectral gap* which accounts for bottlenecks, and *effective resistance* which takes into account alternative pathways and their length, may be two important measures for robustness analysis.

Scenario 2: Water Distribution Networks. Cities and municipalities depend on a mixture of complex and interconnected infrastructure to provide a reliable and safe source of water to consumers. A serious concern for the water utilities providing this service is the vulnerability of water distribution networks (WDNs) to common failures (e.g., aging pipes) and targeted attacks (e.g., disrupted service) [107, 135]. Due to WDNs natural graph representation, graph robustness measures have become a critical tool in the analysis of network vulnerability. In order to address the concern of vulnerability, multiple criteria have been proposed to evaluate a WDNs robustness:

- (i) *alternative pathways* of supply [108]
- (ii) *bottlenecks* or articulation points [80]
- (iii) *connectedness* of the network [80]
- (iv) *loop redundancy* [107]

Interestingly, compared to transportation networks, WDNs currently use a mixture of graph and spectral approaches to account for these desired properties. Common analysis tools include average distance, diameter and clustering coefficient [107], which could be used to identify graph connectedness and loop redundancy. In addition spectral approaches such as spectral gap and algebraic connectivity have been proposed for use in WDN vulnerability analysis. These techniques could be used to identify bottlenecks and measure the strength of connectivity between subregions [80].

Potential Alternatives. Bottleneck and loop redundancy are important criteria in robustness measure selection. Measures such as the spectral gap and algebraic connectivity currently can account for bottlenecks; while tools like average distance, diameter and clustering coefficient provide a measure of alternative pathways and loop redundancy. However in this case, *natural connectivity* which is intrinsically tied to loop capacity and alternative pathways presents a compelling robustness measure. In addition, *effective resistance* which takes into account alternative pathways and their length could be a strong alternative measure.

2.3 Failures and Targeted Attacks

To understand the underlying mechanisms of network failure and attack, we need to examine the graph properties facilitating these issues. In order to do this, we begin with a brief overview of four classical graph models in Section 2.3.1. This background knowledge on graph models assists in the analysis of network failure and attack in Section 2.3.2 and Section 2.3.3, respectively.

2.3.1 Graph Models

We analyze the properties of graph models since they have been extensively studied and contain well defined properties. In addition, through understanding the robustness of these graph models, which are representative of many real-world datasets, we can use the knowledge gained here and apply it to any type of graph data.

Erdős-Rényi (ER) Model [136] The ER model generates random networks with no particular structural bias, where each graph starts with n vertices and no edges. For each pair

of nodes, an edge is added to the graph with probability p . This leads to the Poisson-type degree distribution where the probability of a vertex having degree k is $p_k = p(n-1)$ [137]. In addition, the ER model has a logarithmically increasing average distance and a clustering coefficient close to zero [63]. Potentially the most important property of Erdős-Rényi graphs in relation to network robustness is the homogeneity of the degree and betweenness distributions. This property of homogeneity implies that node importance and network load are evenly distributed among the nodes in the graph [105]. As we will see in the following sections, this is a particularly valuable property.

Watts-Strogatz (WS) Model [102] The WS model generates graphs with high clustering coefficient and low diameter (small-world property). The model starts by generating a ring lattice with n vertices, where every node is connected to its k nearest neighbors. Each edge is then visited once and rewired with probability p to a vertex chosen uniformly at random (no duplicate edges or self-loops allowed). This has the effect of creating “shortcuts” across the graph, creating the low diameter, small-world property.

As such, the WS model shows a heterogeneous betweenness distribution where a small number of nodes have very large betweenness while most nodes contain very little (not a power law distribution though). However, the WS model does manage to maintain a homogeneous degree distribution due the small number of edges rewired [105].

Barbási-Albert (BA) Scale-Free Model [138] The BA model generates network topology according to two processes—*growth* and *preferential attachment*. Where prior network models kept the number of nodes fixed during the network formation process, the BA model starts with a small set of vertices and *grows* the network by adding nodes and edges over time. The reason why the BA model follows the sociological principle of the “rich get richer”, where the probability of connecting to a node is proportional to the degree of that node, is due to *preferential attachment* mechanism [137]. The BA model generates scale-free networks following a power law degree distribution given by $p_k \approx k^{-3}$ with average geodesic distance increasing logarithmically with the size of n , demonstrating the small-world property [63]. As such, the BA model shows both a heterogeneous degree and betweenness distribution, where a small number of nodes account for large proportion of the betweenness load and degree [105].

Clustered Scale-Free (CSF) Model [139] The CSF model extends the BA model by incorporating a high clustering coefficient through a single step after preferential attachment, called *triad formation*. The advantage of the CSF model compared to the BA model is that it generates graphs with high clustering coefficient while maintaining the scale-free and small average distance properties. As such, the degree distribution of the CSF model is

heterogeneous and roughly equivalent to the BA model [139].

2.3.2 *Isolated & Cascading Failures*

These types of failures in a network often occur when a piece of equipment breaks down due to natural causes. In the study of graphs, this could correspond to the removal of either a node or an edge. While random and cascading failures do occur, they are often less severe than targeted attacks. In fact, [105] shows that cascading failures are significantly less impactful than targeted attacks across a range of graph models—ER, WS and BA. For both the ER and BA graphs, the network damage from cascading failures is minimal. While the WS model suffered significant damage from the cascading failure, it was still significantly less than the targeted attack. As a whole, this indicates that isolated and cascading failures are less threatening than targeted attacks. As such, we focus the majority of our attention to targeted attacks.

2.3.3 *Targeted Attacks*

There are only two ways an adversary can attack a network—*removal of nodes* or *removal of edges*. The goal of the attacker is select nodes and edges considered important to the functionality of the network (e.g., critical power substations or electrical lines). To accomplish this, an attacker typically relies on measures of node and edge centrality. While any centrality measure can be used to generate a list of the top k nodes and edges to remove, we focus on two traditional measures—degree and betweenness centrality. Through these two measure, there are four common attack strategies: initial degree removal (ID), initial betweenness removal (IB), recalculated degree distribution removal (RD) and recalculated betweenness removal (RB) [63]. Each of these attack strategies are discussed below.

Initial Degree Removal (ID) In this attack scenario each node v in the network is ranked according to its degree δ_v . For a given budget k , an attacker removes vertices one by one starting with the highest degree nodes. This has the effect of reducing the total number of edges in the network as fast as possible [63]. Since this attack only considers its' local neighborhood when making a decision, it is considered a *local attack*. The benefit of this locality is that the attack strategy is quick to compute; linear in the size of nodes in the graph. The same strategy can be applied to the removal of edges from the network, where edge degree δ_e can be defined in several ways. Four common ones are shown below [63]:

$$\delta_e = \delta_u \cdot \delta_v \quad (2.28a)$$

$$\delta_e = \delta_u + \delta_v \quad (2.28b)$$

$$\delta_e = \min(\delta_u, \delta_v) \quad (2.28c)$$

$$\delta_e = \max(\delta_u, \delta_v) \quad (2.28d)$$

In practice, Equation 2.28 (a) has been shown to be the most effective method of defining edge degree in relation to attack strategy, since it correlates best with betweenness centrality [63].

Initial betweenness Removal (IB) This attack ranks each node $v \in V$ according to its betweenness centrality b_v in Equation 2.4. The attacker then removes up to k nodes, one at a time in descending order of importance. This has the effect of destroying as many paths as possible [63]. Since this attack considers information from across the network, it is considered a *global attack* strategy. As a result, the IB attack strategy is significantly more computationally expensive than ID (see Section 2.2.1). The same attack strategy can be applied to the removal of edges using the definition of edge betweenness centrality:

Recalculated Degree Removal (RD) Recalculated degree removal follows the same steps as ID, except that it recalculates the degree distribution of nodes after removing each vertex. This allows for the attacker to re-assess the target after each round of attack. The same process holds for removing edges.

Recalculated betweenness Removal (RB) Recalculated betweenness removal follows the same steps as IB, except that it recalculates the betweenness centrality of each node after a link is removed. The same process holds for removing edges.

Comparison of Targeted Attacks

The efficacy of each attack outlined in Section 2.3.3 is discussed in relation to two robustness measures: size of the largest connected component (L) and average inverse distance \bar{d}^{-1} , on four classic graph models—(1) Erdős-Rényi (ER), (2) Watts-Strogatz (WS), (3) Barabási-Albert (BA) scale-free and (4) Clustered Scale-Free Model (CSF). In each section below, we review the efficacy of attacks on each type of graph.

Attacking Erdős-Rényi Graphs. The most effective node attack strategy on ER graphs is RD. However, it takes the removal of $\sim 30\%$ of the most central nodes in the graph in order to reduce the \bar{d}^{-1} by 50%; and the removal of $\sim 40\%$ of the central nodes to decrease L

by 50% [63]. This is a large fraction of the nodes to be removed, indicating that ER graphs are highly robust to targeted attacks. This robustness of Erdős-Rényi graphs stems from the homogeneous degree and betweenness distributions. Since these distributions evenly spread the load and importance among each of the nodes. Therefore if a few nodes are attacked, no serious network damage occurs [105].

The ER model is also robust to edge based attacks. The most effective attack strategy is again RB, where approximately 40% of the edges have to be removed in order to drop the average inverse distance by 50%. To reach a similar performance drop in the largest connected component, approximately 50% of the edges needed to be removed by the most effective edge attack strategy RD [63].

Attacking Watts-Strogatz Graphs. The attack behavior on WS graphs is significantly different from ER graphs. For just a small number of removed vertices, the RB procedure completely breaks down the structure of the graph. By removing just $\sim 2\%$ of the most central nodes, \bar{d}^{-1} is reduced by 45%. After removing roughly 10% of the top nodes, the network structure completely breaks down resulting in a 95% decrease in \bar{d}^{-1} and an 85% decrease in L [63].

However, results from [63] show that degree based strategies are largely ineffective in attacking WS graphs. This largely stems from the fact that WS networks have a heterogeneous betweenness distribution and a homogeneous degree distribution. Since the RB attack targets central nodes carrying a majority of the load, the graph rapidly disintegrates. On the other hand, targeting nodes based on degree distribution is not as effective since most nodes have roughly the same degree [105].

Evaluating the robustness of WS graphs under edge attacks results in similar performance degradation compared to node based ones. Again, the RB attack effectively deconstructs the graph after the removal of only 20% of the edges, reducing L to $< 5\%$ of its original value. In addition, the average inverse distance drops to roughly 10% of the original value at the same mark. The effectiveness of the RB strategy can be tied to the identification of important edges, which in the WS model are the rewired edges linking the distant parts of the ring [63].

Attacking Barabási-Albert Scale-Free Graphs. The BA graph is significantly more robust to node based attacks than the WS model, even though the BA graph has a heterogeneous degree *and* betweenness distribution. While this may seem odd at first, it likely stems from the fact that the WS model is based on a ring structure with mostly local connections. Therefore, removal of even a single node significantly affects the neighborhood [105]. Compared to the ER model, the BA graph suffers significantly from both RB and RD attacks due to the heterogeneous nature of the degree and betweenness distribution.

Edge based attacks on the BA model perform poorly compared to their node based counterparts. It takes a removal of 40% of the edges in order to drop the performance of \bar{d}^{-1} by 50%. In comparison, it would have only the removal of $\sim 12\%$ of the nodes to reach the same level of performance drop. Similar results are found when comparing the largest connected component performance across node and edge attacks [63]. One potential reason for this significant difference could arise from the combination of the homogeneous nature of the betweenness distribution and the hub-like structure of important vertices.

Attacking Clustered Scale-Free Graphs. The CSF model is an extension of BA with the addition of a triad step, allowing the model to generate graphs with a higher clustering coefficient. However, it turns out that this clustering step introduces significant vulnerability into the network. This vulnerability is likely caused by the targeting of important vertices with many triads attached to it [63]. To put it in comparison, removing just 10% of the central nodes from the CSF graph results in both L and \bar{d}^{-1} dropping by 90%. In comparison, the BA graph which is widely known to be vulnerable to targeted attacks, only drops roughly 50% in L and 10% in \bar{d}^{-1} with the same number of nodes removed [63].

Similar to the BA graph, the CSF graph seem to be much more resilient to edge based attacks than node based ones [63]. However, it is still significantly more vulnerable to RD based edge attacks than the BA model.

2.3.4 Comparison to Other Targeted Attacks

While we have studied two types of centrality based attacks on networks—node centrality and betweenness centrality—there exists a number of alternative options [34]. Some of these alternatives include: PageRank [140], closeness centrality [141], eigenvector centrality and Katz centrality [142]. However, it has been shown that many of these centrality scores produce highly correlated results when conducting targeted attacks on networks [34]. The three most common groupings of centrality measures according to similarity are as follows: (PageRank, betweenness centrality), (Katz centrality, eigenvector centrality) and (closeness centrality, degree centrality). As such, attack related studies may want to consider evaluating attacks from distinct groups in order to avoid similar attack patterns.

2.4 Network Defense

To understand the best mechanism of defense for a particular network, it is important to understand the properties of the graph and the type of attack or failure we are expecting

to protect against. In Section 2.4.1 we overview measure independent heuristics for improving network defense, grouping each technique into one of three categories depending on whether it improves network robustness through (i) edge addition, (ii) edge rewiring, or (iii) identifies important nodes and edges in a network to monitor suspicious activity. Then in Section 2.4.2 we analyze optimization based techniques for network defense according to the same categorization process as above. Finally, in Section 2.4.3 we discuss when to apply different defense techniques.

2.4.1 Measure Independent Heuristics

In this section we evaluate network defenses that are heuristic in nature. This means that the technique modifies the graph structure independent of a robustness measure [26].

Edge Addition

Edge additions typically incur additional network costs than edge rewiring, however, they almost always provide better results. In [23], they show that edge addition outperforms all proposed edge rewiring schemes on two measures of network robustness L and \bar{d}^{-1} . In addition, they find that not all edge addition techniques are equal, and that preferential edge addition outperforms random edge addition.

Edge Rewiring

Edge rewiring is a popular technique to improve network robustness since it generally has a lower cost associated with it compared to adding edges. As such, we study the six methods of edge rewiring proposed in [23]:

- (1) *Random addition*: add an edge connecting two random nodes
- (2) *Preferential addition*: add an edge by connecting two nodes having the lowest degrees
- (3) *Random edge rewiring*: remove a random edge, then add an edge as in (1)
- (4) *Random neighbor rewiring*: randomly select a node, and then a random neighbor of that node, then remove the corresponding edge. Next add an edge as in (1)
- (5) *Preferential rewiring*: disconnect a random edge from the highest-degree node and reconnect that edge to a random node
- (6) *Preferential random edge rewiring*: choose a random edge, disconnect it from the higher degree node, then connect that edge to a random node

In order to evaluate the effectiveness of each proposed rewiring scheme, [23] proposes to measure network robustness through the average inverse distance \bar{d}^{-1} and largest connected component L on three increasing levels of node attack. With respect to robustness measure L , the conclusion that was drawn is that edge rewiring is most effective in the following order: 5 > (3 and 6) > 4. Each number corresponds to the list above. On the other hand when using \bar{d}^{-1} is used as the metric, preferential rewiring (6) is found to increase \bar{d}^{-1} the most for small amounts of rewiring, while random edge (3) and random neighbor (4) techniques give the largest \bar{d}^{-1} improvement when large numbers of edges were rewired [23].

Node Monitoring

Many of the centrality based techniques that are used to attack a network can be used to defend it. In fact, if you know that an adversary is using one of these approaches to attack your network, you can monitor the same nodes in advance. To this end, a range of centrality measures can be used for node monitoring, including: degree centrality, betweenness centrality, PageRank, closeness centrality, eigenvector centrality, Katz centrality, etc. While picking the appropriate measure to monitor your network might seem overwhelming, it has been shown that a high degree of correlation exists between many of these measures. The three most common groupings of centrality measures are: (PageRank, betweenness centrality), (Katz centrality, eigenvector centrality) and (closeness, degree) [34]. This can help reduce the burden of selecting centrality measures for defense by eliminating redundant and highly correlated measures.

2.4.2 Optimization Based Techniques

The focus of optimization based methods is to modify the underlying graph structure through the manipulation of a targeted robustness measure [26].

Edge Rewiring

For this section, we focus our attention to the optimization based method for edge rewiring proposed in [26]. In this work, they propose an algorithm EDGEREWIRE that maximizes a particular spectral robustness measure according to a given budget. In addition, they only allow modifications based on degree-preserving edge rewirings in order to ensure that load on nodes remains unchanged. In total, they allow EDGEREWIRE to operate on six spectral measures, including: spectral radius, spectral gap, natural connectivity, algebraic connectivity, effective resistance and number of spanning trees.

Comparing EDGEREWIRE to a series of heuristic based edge rewiring approaches on 14 datasets, they find that the proposed method significantly outperforms heuristic based approaches when measuring for the optimized spectral parameter.

Node Monitoring

We focus our study of optimization based node monitoring techniques to the work conducted by Tong et. al. [5]. In this work, they propose a three step process for network defense against virus propagation—(i) evaluation of a graphs’ vulnerability to virus propagation via the spectral radius; (ii) design of the ‘Shield-value’ measure to quantify the importance of a set of nodes in protecting the graph; and (iii) development of a quick algorithm utilizing the Shield-value measure to determine the k best nodes to protect the graph.

The spectral radius λ_1 was chosen as a natural measure of graph robustness to virus propagation due to its close link to the epidemiological threshold. As such, in order to minimize the spread of a virus on a network the goal is to minimize λ_1 by selecting the best set S of k nodes to remove from the graph (i.e., maximize eigendrop). In order to evaluate the goodness of a node set S for removal, [5] proposes the Shield-value measure:

$$Sv(S) = \sum_{i \in S} 2\lambda_1 \mathbf{u}_1(i)^2 - \sum_{i,j \in S} A(i,j) \mathbf{u}(i) \mathbf{u}(j) \quad (2.29)$$

The intuition behind this equation is that we want to select nodes for removal/monitoring that have high eigenvector centrality (first term) while penalizing nodes for being close together (second term). In order to select this set of nodes S , they develop the NetShield algorithm. We refer the reader to [5] for technical details. To evaluate the efficacy of the node monitoring approach, it is compared across a multitude of heuristic based measures, including— PageRank, degree centrality, etc.—finding that the NetShield approach to node monitoring outperforms traditional centrality based approaches in mitigating the spread of viruses on a network.

2.4.3 Selecting a Defense Method

The most devastating attacks often correspond to targeted attacks, rather than isolated or cascading failures [105]. As a result, it is common for defense mechanisms to prioritize the protection of networks from targeted attacks, unless prior information indicates otherwise. Since an attacker likely targets nodes or edges based on a measure of centrality, we have information *a priori* on our adversary. As the defender, we often have knowledge of the

graph topology and underlying degree distribution. This allows us to better quantify our robustness and identify potential points of attack. For example, we can measure the number of bottlenecks present in our network; along with the availability of alternative pathways. Perhaps most importantly, we can determine the degree and betweenness distribution of the graph topology, allowing us to make additional assumptions about the best defense strategy.

Since many real world networks assume a heterogeneous degree distribution, where a few nodes contain many links and many nodes contain only a few links; we use it as an example network to defend. In this scenario we can: (1) monitor critical nodes according to different measures of centrality (e.g., nodes with many links); (2) rewire the graph in an attempt to increase robustness; or (3) add edges to the network in order to increase robustness. In practice, these decisions often depend on the specific application domain and the cost of the associated action. However, the most cost effective measure is arguably node monitoring, which if implemented carefully, can prevent targeted attacks from ever occurring.

2.5 Research Directions & Open Problems

We present research directions and open problems distilled from the surveyed works. Four promising directions include: (1) an axiomatic study of desired properties in robustness measures, helping guide the selection and development of new measures; (2) interpretability of robustness measures to assist users in understanding the impact of measure scores; (3) applying the study of network robustness to additional high-impact domains such as physical security and cybersecurity; and (4) bridging the study of graph vulnerability and robustness with recent developments in adversarial machine learning on graph structured data.

2.5.1 Guidelines for Selecting & Developing Measures

Comparing robustness measures in a quantitative manner is still an open challenge. While many works have qualitatively remarked on why certain robustness measures are better suited for certain tasks, there has been no formal study outlining desirable characteristics that a robustness measure should contain. By formalizing these desirable properties into a set axioms, future and existing robustness measures could be compared in an independent and quantitative manner, something that is not currently available. We identified 6 desirable robustness properties across the literature that could form the basis for an axiomatic analysis of robustness measures [25, 26, 47]. Below, we provide the intuition for each axiom, however, each axiom needs to be formalized and (dis)proven for each robustness measure.

- 1. Strictly Monotonic.** When an edge is added to a graph the network connectivity is intrinsically enhanced. A robustness measure should account for this increased connectivity by strictly increasing (or decreasing) for each edge added to the graph.
- 2. Redundancy.** A critical ability of any robustness measure is to measure redundancy present in the network. This means that if multiple paths between two nodes exist, the proposed measure should be able to account for both the number of paths and their quality (where smaller paths are better).
- 3. Disconnected.** Many real-world graphs contain disconnected components; therefore a measure should be able to evaluate a graphs' robustness independent of the number of disconnected components.
- 4. Stable.** A robustness measure should change in proportion to the perturbation of the graph structure. For example, if a single edge is added to a graph, we expect that the measure has a proportionally small response.
- 5. Consistent.** Given two graphs with same underlying structure, we would expect them to have similar robustness independent of their size.
- 6. Scalable.** Large graphs containing millions (or sometimes billions) of nodes and edges are common. A robustness measure should be scalable to large graphs, where we define scalable as an algorithm subquadratic with respect to the number of nodes and edges.
- 6. Intuitive.** Ideally, we want robustness measures to have identifiable connections to the underlying graph topology, and for these connections to be conveyable to non-experts in an understandable manner.

2.5.2 Furthering Interpretability

Ideally, robustness measure should have identifiable connections to the underlying graph topology to explain what the robustness score is indicative of. Recent research has explored this in the more general domains of graph connectivity and ranking [143, 144, 145, 146] Combining visual representations, helpful interactions, and state-of-the-art attribution and feature visualization techniques together into rich user interfaces could lead to major breakthroughs in understanding graph vulnerability and robustness scores.

2.5.3 Studying Robustness in New Domains

The study of graph vulnerability and robustness is still nascent in the areas of physical security [44], cybersecurity [59] and interdependent and dynamic networks [147]. For physical security, [44] studies the vulnerability and robustness of physical sensor placement to

maximize perimeter security while minimizing network latency. They find that perimeter security systems frequently map to circular lattices which suffer from a trade-off between robustness and mean path length (i.e., latency). Future work could analyze alternative perimeter system mappings that optimize for both criteria, while exploring alternative definitions of robustness in physical security. With respect to cybersecurity, [59] attempts to calculate the vulnerability and robustness of enterprise networks by modeling lateral attack movement between computers. However, their unique probabilistic robustness measure is dependent on cyber domain knowledge and the running of many simulations. Future cybersecurity robustness analysis could explore the development of robustness measures that are simulation independent, reducing computational costs and the need for explicit domain knowledge. Many real-world networks are often dynamic and contain multiple interdependent sub-networks. While initial work has looked at real-time robustification of interdependent networks from an edge perspective [147], additional work needs to be done to (i) study dynamic graphs, (ii) comprehensively evaluate various attack and defense scenarios, and (iii) develop unique robustness measures that can better account for the nature of inter-dependent and dynamic networks.

2.5.4 Bridging Graph Robustness & Adversarial ML

From the machine learning perspective, a majority of current graph robustness research focuses on manipulating graph classifiers or embedding mechanisms into mispredicting the label of a graph [148], or the label of each node in the graph [149]. So far, adversarial machine learning research has yet to deeply delve into the more richly defined network robustness objective centered around a networks' ability to continue functioning when damaged or attacked. However, we believe that there are multiple high-impact connections to explore, including: (1) how does a graph's spectral robustness (e.g., spectral gap) correlate to the vulnerability or robustness of downstream tasks such as node and graph classifiers being attacked; and (2) does optimizing a graph's spectral robustness (e.g., adding or rewiring edges) affect an attackers ability to perturb downstream node and graph classification models. By answering these questions, we can uncover new mechanisms to attack and defend networks while gaining insight into fundamental connections between two important and growing fields.

2.6 Conclusion

In this survey, we distill answers to key questions that are currently scattered across multiple scientific fields and numerous papers. In particular, we provide researchers and practi-

tioners crucial access to network robustness information by—(1) summarizing and comparing 18 recent and classical graph robustness measures; (2) exploring which robustness measures are most applicable to different categories of network data (e.g., social, infrastructure); (3) reviewing common network attack strategies, and summarizing which attacks are most effective across different network topologies; and (4) extensive discussion on selecting defense techniques to mitigate attacks across a variety of networks. We concluded by highlighting current research directions and open problems. This survey will serve as a guide to researchers and practitioners in navigating the expansive field of network robustness, while summarizing answers to key questions.

CHAPTER 3

EVALUATING GRAPH VULNERABILITY AND ROBUSTNESS USING TIGER

Network robustness plays a crucial role in our understanding of complex interconnected systems such as transportation, communication, and computer networks. While significant research has been conducted in the area of network robustness, no comprehensive open-source toolbox currently exists to assist researchers and practitioners in this important topic. This lack of available tools hinders reproducibility and examination of existing work, development of new research, and dissemination of new ideas. We contribute TIGER, an open-sourced Python toolbox to address these challenges. TIGER contains 22 graph robustness measures with both original and fast approximate versions; 17 failure and attack strategies; 15 heuristic and optimization-based defense techniques; and 4 simulation tools. By democratizing the tools required to study network robustness, our goal is to assist researchers and practitioners in analyzing their own networks; and facilitate the development of new research in the field. TIGER has been integrated into the Nvidia Data Science Teaching Kit available to educators across the world; and Georgia Tech’s Data and Visual Analytics class with over 1,000 students. TIGER is open sourced at: <https://github.com/safreital/TIGER>.

3.1 Introduction

Through analyzing and understanding the robustness of networks we can: (1) quantify network vulnerability and robustness, (2) augment a network’s structure to resist attacks and recover from failure, and (3) control the dissemination of entities on the network (e.g., viruses, propaganda). Consider the impactful scenario where a virus penetrates one or more machines in an enterprise network. Once infected, the virus laterally spreads to susceptible machines in the network, resulting in system-wide failures, data corruption and exfiltration of trade secrets and intellectual property. This scenario is commonly modeled as a dissemination of entities problem using an epidemiological susceptible-infected-susceptible (SIS) model, where each machine is in either one of two states—infected or susceptible (see Figure 3.1). How quickly a virus spreads across a network is known as the network’s **vulnerability**, and is defined as a *measure of susceptibility to the dissemination of entities across the network* [5]. A natural counterpart to network vulnerability is **robustness**, defined as a *measure of a network’s ability to continue functioning when part of the network is naturally damaged or targeted for attack* [25, 26, 23]

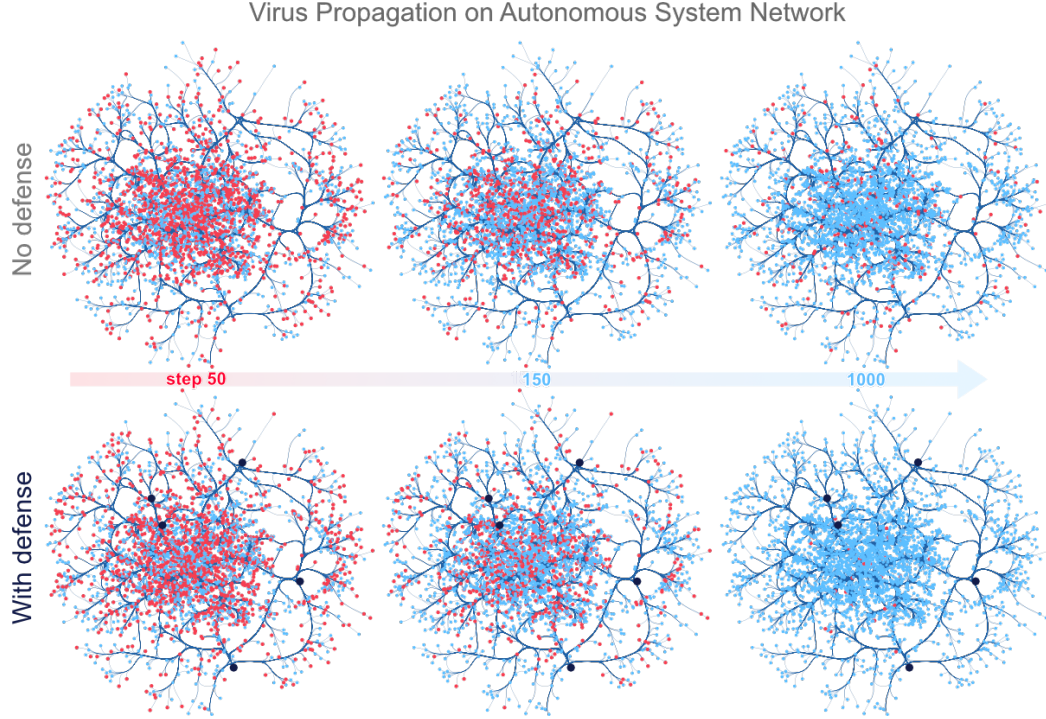


Figure 3.1: TIGER provides a number of important tools for graph vulnerability and robustness research, including graph robustness measures, attack strategies, defense techniques and simulation models. Here, a TIGER user is visualizing a computer virus simulation that follows the SIS infection model (effective strength $s = 3.21$) on the *Oregon-1 Autonomous System* network [4]. Top: without any defense, the virus remains endemic. Bottom: defending only 5 nodes with *Netshield* [5], the number of infected entities is reduced to nearly zero.

Challenges for robustness and vulnerability research. Network robustness has a rich and diverse background spanning numerous fields of engineering and science [22, 23, 5, 24, 59]. Unfortunately, this cross-disciplinary nature comes with significant challenges—resulting in slow dissemination of ideas, leading to missed innovation opportunities. We believe a unified and easy-to-use software framework is key to standardizing the study of network robustness, helping accelerate reproducible research and dissemination of ideas.

TIGER design and implementation. We present TIGER, an open-sourced Python *Toolbox for evaluatIng Graph vulnErability and Robustness*. Through TIGER, our goal is to catalyze network robustness research, promote reproducibility and amplify the reach of novel ideas. In designing TIGER, we consider multiple complex implementation decisions, including: (1) the criterion for inclusion in the toolbox; (2) identifying and synthesizing a set of core robustivity features needed by the community; and (3) the design and implementation of the framework itself. We address the *inclusion criterion* by conducting a careful analysis of influential and representative papers (e.g., [26, 5, 74, 63, 23]) across top journals and conferences from the relevant domains (e.g., ICDM, SDM, Physica A, DMKD, Phys-

ical Review E), many of which we will discuss in detail in this chapter. We also include papers posted on arXiv, as many cutting-edge papers are first released there.

Based on our analysis, we identify and include papers that tackle one or more of the following fundamental tasks [25, 63, 23]: (1) measuring network robustness and vulnerability; (2) understanding network failure and attack mechanisms; (3) developing defensive techniques; and (4) creating simulation tools to model processes. From these papers, we select and implement a total of 44 attacks, defenses and robustness measures, along with 4 simulation tools in which they can be used. Due to a vibrant and growing community of users, we develop TIGER in Python 3, leveraging key libraries, such as NetworkX, SciPy, Numpy and Matplotlib. While excellent alternative network analysis tools exist [150, 151, 152, 153, 145, 154, 155, 156], many of them are domain specific (e.g., EoN [151], WNTR [153]) or do not provide direct support for network robustness analysis (e.g., NetworkX [150], Gephi [156]). In contrast, TIGER complements existing tools while providing key missing network robustness components.

3.1.1 Contributions

1. TIGER. We present TIGER, the first open-sourced Python toolbox for evaluating network vulnerability and robustness of graphs. TIGER contains 22 graph robustness measures with both original and fast approximate versions when possible; 17 failure and attack mechanisms; 15 heuristic and optimization based defense techniques; and 4 simulation tools. TIGER also supports a large number of GPU accelerated robustness measures. To maintain the integrity of the code base, TIGER uses continuous integration to run a suite of test cases on every commit. To the best of our knowledge, this makes TIGER the most comprehensive open-source framework for network robustness analysis to date.

2. Open-Source & Permissive Licensing. Our goal is to democratize the tools needed to study network robustness; assisting researchers and practitioners in the analysis of their own networks. As such, we open-source the *code* on Github and PyPi with an *MIT license* available at: <https://github.com/safreital/TIGER>.

3. Extensive Documentation & Tutorials. We extensively document the functionality of TIGER, providing docstrings for each function and class, along with quick examples on how to use the robustness measures, attacks, defenses, and simulation frameworks. In addition, we provide 5 detailed tutorials—one for every major component of TIGER’s functionality—on multiple large-scale, real-world networks, including *every* figure and plot shown in this chapter. Users with Python familiarity will be able to readily pick up TIGER for analysis with their own data.

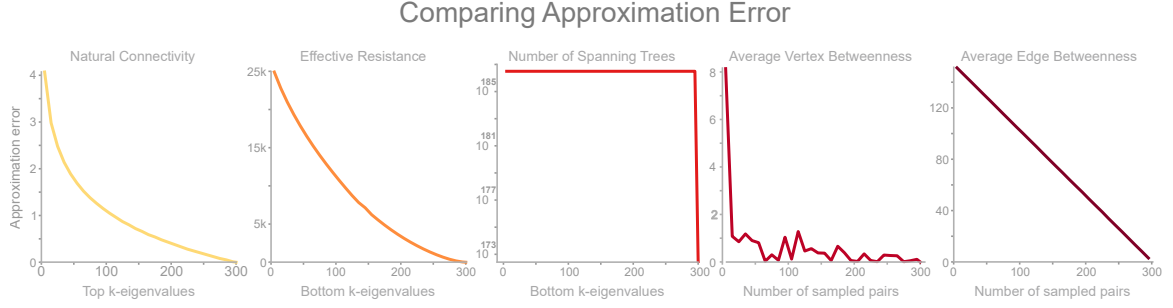


Figure 3.2: Error of 5 fast, approximate robustness measures supported by TIGER. Parameter k represents the trade-off between speed (low k) and precision (high k). To measure approximation efficacy, we vary $k \in [5, 300]$ in increments of 10 and measure the error between the approximate and original measure averaged over 30 runs on a clustered scale-free graph with 300 nodes.

4. Community Impact. TIGER helps enable reproducible research and the timely dissemination of new and current ideas in the area of network robustness and vulnerability analysis. As part of the newly released Nvidia Data Science Teaching Kit, TIGER will be used by educators and researchers across the world. TIGER has been integrated into the Nvidia Data Science Teaching Kit available to educators across the world; and Georgia Tech’s Data and Visual Analytics with over 1,000 students. Since this is a *large* and *highly active* field across many disciplines of science and engineering, we anticipate that TIGER will have significant impact. As the field grows, we will continue to update TIGER with new techniques and features.

3.2 TIGER Robustness Measures

TIGER contains 22 robustness measures, grouped into one of three categories depending on whether the measure utilizes the graph, adjacency, or Laplacian matrix. We present 3 representative robustness measures, one from each of the three categories, to extensively discuss. For detailed description and discussion of all 22 measures, we refer the reader to the online documentation.

Terminology and Notation. As the study of graphs has been carried out in a variety of fields (e.g., mathematics, physics, computer science), the terminology often varies from field to field. As such, we refer to the following word pairs interchangeably: (network, graph), (vertex, node), (edge, link). Throughout the chapter, we follow standard practice and use capital bold letters for matrices (e.g., \mathbf{A}), lower-case bold letters for vectors (e.g., \mathbf{a}). Also, we focus on undirected and unweighted graphs.

3.2.1 Example Measures

Average vertex betweenness (\bar{b}_v) of a graph $G = (\mathcal{V}, \mathcal{E})$ is the summation of vertex betweenness b_u for every node $u \in V$, where vertex betweenness for node u is defined as the number of shortest paths that pass through u out of the total possible shortest paths

$$\bar{b}_v = \sum_{u \in V} \sum_{s \in V} \sum_{\substack{t \in V \\ s \neq t \neq u}} \frac{n_{s,t}(u)}{n_{s,t}} \quad (3.1)$$

where $n_{s,t}(u)$ is the number of shortest paths between s and t that pass through u and $n_{s,t}$ is the total number of shortest paths between s and t [120]. Average vertex betweenness has a natural connection to graph robustness since it measures the average load on vertices in the network. The smaller the average the more robust the network, since load is more evenly distributed across nodes.

Spectral scaling (ξ) indicates if a network is simultaneously sparse and highly connected, known as “good expansion” (GE) [127, 57]. Intuitively, we can think of a network with GE as a network lacking bridges or bottlenecks. In order to determine if a network has GE, [57] proposes to combine the spectral gap measure with odd subgraph centrality SC_{odd} , which measures the number of odd length closed walks a node participates in. Formally, *spectral scaling* is described in Equation 3.2,

$$\xi(G) = \sqrt{\frac{1}{n} \sum_{i=1}^n \{ \log[\mathbf{u}_1(i)] - [\log \mathbf{A} + \frac{1}{2} \log[SC_{odd}(i)]] \}^2} \quad (3.2)$$

where $\mathbf{A} = [\sinh(\lambda_1)]^{-0.5}$, n is the number of nodes, and \mathbf{u}_1 is the first eigenvector of adjacency matrix \mathbf{A} . The closer ξ is to zero, the better the expansion properties and the more robust the network. Formally, a network is considered to have GE if $\xi < 10^{-2}$, the correlation coefficient $r < 0.999$ and the slope is 0.5.

Effective resistance (R) views a graph as an electrical circuit where an edge (i, j) corresponds to a resistor of $r_{ij} = 1$ Ohm and a node i corresponds to a junction. As such, the effective resistance between two vertices i and j , denoted R_{ij} , is the electrical resistance measured across i and j when calculated using Kirchoff’s circuit laws. Extending this to the whole graph, we say the *effective graph resistance* R is the sum of resistances for all distinct pairs of vertices [25, 62]. Klein and Randic [22] proved this can be calculated based on the sum of the inverse non-zero Laplacian eigenvalues:

$$R = \frac{1}{2} \sum_{i,j} R_{ij} = n \sum_{i=2}^n \frac{1}{\mu_i} \quad (3.3)$$

As a robustness measure, effective resistance measures how well connected a network is, where a smaller value indicates a more robust network [62, 25]. In addition, the effective resistance has many desirable properties, including the fact that it strictly decreases when adding edges, and takes into account both the number of paths between node pairs and their length [56].

3.2.2 Measure Implementation & Evaluation

Our goal for TIGER is to implement each robustness measure in a clear and concise manner to facilitate code readability, while simultaneously optimizing for execution speed. Each robustness measure is wrapped in a function that abstracts mathematical details away from the user; and any default parameters are set for a balance of *speed* and *precision*. Below we compare the efficacy of 5 fast, approximate robustness measures, followed by an analysis of the scalability of all 22 measures.

Approximate Measures. It turns out that a large number of robustness measures have difficulty scaling to large graphs. To help address this, we implement and compare 5 fast approximate measure, three spectral based (natural connectivity, number of spanning trees, effective resistance), and two graph based (average vertex betweenness, average edge betweenness) [26, 157]. To approximate *natural connectivity* we use the top- k eigenvalues of the adjacency matrix as a low rank approximation [26, 74]. For the *number of spanning trees* and *effective resistance* we take the bottom- k eigenvalues of the Laplacian matrix [26]. For graph measures, *average vertex betweenness* and *average edge betweenness*, we randomly sample k nodes to calculate centrality. In both cases, the parameter k represents the trade-off between speed (low k) and precision (high k). When k is equal to the number of nodes n in the graph, the approximate measure is equivalent to the original.

To determine the efficacy of each approximation measure we vary $k \in [5, 300]$ in increments of 10, and measure the absolute error between the approximate and original measure, averaged over 30 runs on a clustered scale free graph containing 300 nodes. In Figure 3.2, we observe that average vertex betweenness accurately approximates the original measure using $\sim 10\%$ of the nodes in the graph. This results in a significant speed-up, and is in line with prior research [157]. While the absolute error for each spectral approximation is large, these approximations find utility in measuring the relative change in graph robustness after a series of perturbations (i.e., addition or removal of nodes/edges). While not immediately obvious, this can enable the development a wide range of optimization based defense techniques [26, 47].

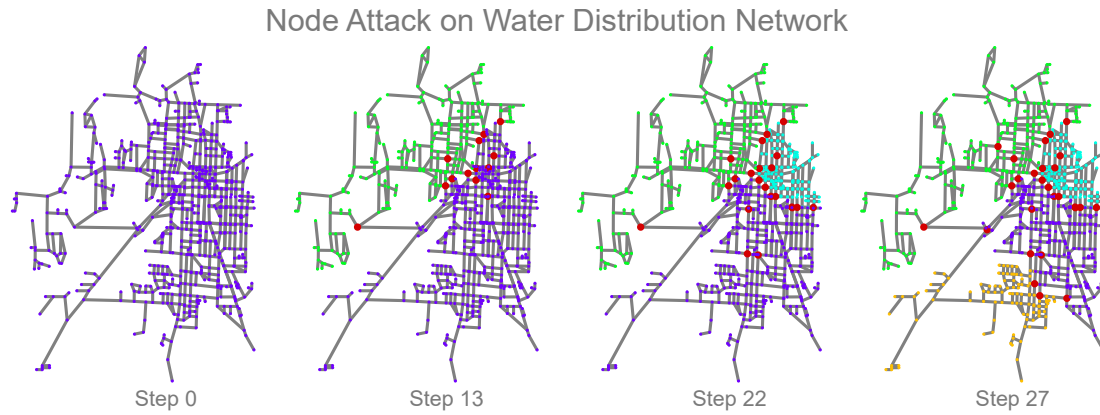


Figure 3.3: TIGER simulation of an RD node attack on the KY-2 water distribution network. Step 0: network starts under normal conditions; at each step a node is removed by the attacker (red nodes). Step 13, 22 & 27: after removing only a few of the 814 nodes, the network splits into two and three and four disconnected regions, respectively.

3.2.3 Running Robustness Measures in TIGER

The code block in Listing 1 illustrates how TIGER abstracts the code complexity away from the user, enabling them to quickly evaluate the robustness of their own network data in a simple manner. In line 1, we import a helper function to generate various NetworkX graphs; line 2 imports a utility function to run the specified robustness measure; line 5 creates a Barabasi-Albert (BA) graph with 1000 nodes; and in lines 8 and 12 we calculate the graph's spectral radius and effective resistance, respectively.

```

1 from graph_tiger.graphs import graph_loader
2 from graph_tiger.measures import run_measure
3
4 # Load a Barabasi-Albert graph with 1000 nodes
5 graph = graph_loader(graph_type='BA', n=1000, seed=1)
6
7 # Calculate graph's spectral radius
8 sr = run_measure(graph, measure='spectral_radius')
9 print("Spectral radius:", sr)
10
11 # Calculate graph's effective resistance
12 er = run_measure(graph, measure='effective_resistance')
13 print("Effective resistance:", er)

```

Listing 3.1: Measuring the spectral radius and effective resistance of a Barabasi-Albert (BA) graph using TIGER

3.3 TIGER Attacks

There are two primary ways a network can become damaged—(1) *natural failure* and (2) *targeted attack*. Natural failures typically occur when a piece of equipment breaks down from natural causes. In the study of graphs, this would correspond to the removal of a node or edge in the graph. While random network failures regularly occur, they are typically less severe than targeted attacks. This has been shown to be true across a range of graph structures [105, 23]. In contrast, targeted attacks carefully select nodes and edges in the network for removal in order to maximally disrupt network functionality. As such, we focus the majority of our attention to targeted attacks. In Section 3.3.1, we provide a high-level overview of several network failure and attack strategies. Then, in Section 3.3.2 we highlight 10 attack strategies implemented in TIGER.

3.3.1 Attack Strategies

We showcase an example attack in Figure 3.3 on the Kentucky KY-2 water distribution network [158]. The network starts under normal conditions (far left), and at each step an additional node is removed by the attacker (red nodes). After removing only 13 of the 814 nodes, the network is split into two separate regions. By step 27, the network splits into four disconnected regions. In this simulation, and in general, attack strategies rely on node and edge centrality measures to identify candidates. Below, we highlight several attack strategies [63] contained in TIGER.

Initial degree removal (ID) targets nodes with the highest degree δ_v . This has the effect of reducing the total number of edges in the network as fast as possible [63]. Since this attack only considers its neighbors when making a decision, it is considered a *local attack*. The benefit of this locality is low computational overhead.

Initial betweenness removal (IB) targets nodes with high betweenness centrality b_v . This has the effect of destroying as many paths as possible [63]. Since path information is aggregated from across the network, this is considered a *global attack* strategy. Unfortunately, global information comes with significant computational overhead compared to a local attacks.

Recalculated degree (RD) and **betweenness removal (RB)** follow the same process as *ID* and *IB*, respectively, with one additional step to recalculate the degree (or betweenness) distribution after a node is removed. This recalculation often results in a stronger attack, however, recalculating these distributions adds a significant amount of computational overhead to the attack.

3.3.2 Comparing Strategies

To help TIGER users determine the effectiveness of attack strategies, we evaluate 5 node and 5 edge attacks on the Kentucky KY-2 water distribution network in Figure 3.4. We begin by analyzing each node attack strategy—*ID*, *RD*, *IB*, *RB* and *RND* (random selection)—on the left-side of Figure 3.4. Attack success is measured based on how fractured the network becomes when removing nodes from the network. We identify three key observations—(i) random node removal (*RND*) is not an effective strategy on this network structure; (ii) *RB* is the most effective attack strategy; and (iii) the remaining three attacks are roughly equivalent, falling somewhere between *RND* and *RB*.

Analyzing Figure 3.3, we can gain insight into why *RB* is the most effective of the attacks. If we look carefully, we observe that certain nodes (and edges) in the network act as key bridges between various network regions. As a result, attacks able to identify these bridges are highly effective in disrupting this network. In contrast, degree based attacks are less effective, likely due to the balanced degree distribution. The analysis is similar for edge based attacks.

3.3.3 Running Network Attacks in TIGER

The code block in Listing 2 illustrates how TIGER users can quickly run a network attack by modifying 3 parameters—(1) the number of attack simulations ‘runs’, (2) the number of nodes to remove ‘steps’, and (3) the attack strategy ‘attack’. The output of the simulation is a plot of graph robustness (e.g., largest connected component by default) versus attack strength.

```

1 from graph_tiger.attacks import Attack
2 from graph_tiger.graphs import graph_loader
3

```

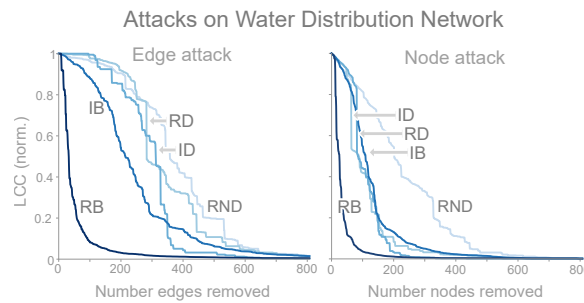


Figure 3.4: Efficacy of 5 edge attacks (left) and 5 node attacks (right) on the KY-2 water distribution network. The most effective attack (*RB*) disconnects approximately 50% of the network with less than 30 removed edges (or nodes).

```

4 params = {
5     'runs': 1,           # number of simulations
6     'steps': 30,         # remove 1 node per step
7     'attack': 'rd_node', # specify attack
8     'seed': 1,           # reproducibility
9 }
10
11 # Load Kentucky KY-2 water distribution network
12 graph = graph_loader(graph_type='ky2')
13
14 # Run and plot attack simulation
15 a = Attack(graph, **params)
16 results = a.run_simulation()
17 a.plot_results(results)

```

Listing 3.2: Attacking the Kentucky KY-2 water distribution network using TIGER

3.4 TIGER Defenses

The same centrality measures effective in attacking a network are important to network defense (e.g., degree, betweenness, PageRank, eigenvector, etc.). In fact, if an attack strategy is known a priori, node monitoring can largely prevent an attack altogether. In Section 3.4.1, we provide a high-level overview of several heuristic and optimization based defense techniques. Then, in Section 3.4.2 we show TIGER users how several defense techniques can be used to robustify an attacked network.

3.4.1 Defense Strategies

We categorize defense techniques based on whether they operate heuristically, modifying graph structure independent of a robustness measure, or by optimizing for a particular robustness measure [26]. Within each category a network can be defended i.e., improve its robustness by—(1) *edge rewiring*, (2) *edge addition*, or (iii) *node monitoring*. Edge rewiring is considered a *low* cost, *less* effective version of edge addition. On the other hand, edge addition almost always provides stronger defense [23]. Node monitoring provides an orthogonal mechanism to increase network robustness by monitoring (or removing) nodes in the graph. This has an array of applications, including: (i) preventing targeted attacks, (ii) mitigating cascading failures, and (iii) reducing the spread of network entities. Below, we highlight several heuristic and optimization based techniques contained in TIGER.

Heuristic Defenses. We overview 5 edge rewiring and addition defenses [23], and compare the effectiveness of them in Section 3.4.2:

1. *Random addition*: adds an edge between two random nodes.
2. *Preferential addition*: adds an edge connecting two nodes with the lowest degrees.
3. *Random edge rewiring*: removes a random edge and adds one using (1).
4. *Random neighbor rewiring*: randomly selects neighbor of a node and removes the edge. An edge is then added using (1).
5. *Preferential random edge rewiring*: selects an edge, disconnects the higher degree node, and reconnects to a random one.

Optimization Defenses. We discuss the Netshield node monitoring technique which identifies key nodes in a network to reduce the spread of entity dissemination (e.g., viruses) [5]. To minimize the spread of entities, Netshield minimizes the spectral radius of the graph λ_1 by selecting the best set S of k nodes to remove from the graph (i.e., maximize eigendrop). In order to evaluate the goodness of a node set S for removal, [5] proposes the Shield-value measure:

$$Sv(S) = \sum_{i \in S} 2\lambda_1 \mathbf{u}_1(i)^2 - \sum_{i,j \in S} A(i,j) \mathbf{u}(i) \mathbf{u}(j) \quad (3.4)$$

The intuition behind this equation is to select nodes for monitoring that have high eigenvector centrality (first term), while penalizing neighboring nodes to prevent grouping (second term). We demonstrate the utility of this defense mechanism in Section 3.5.

3.4.2 Comparing Strategies

To help users evaluate the effectiveness of defense techniques, we compare 5 edge defenses on the Kentucky KY-2 water distribution network, averaged over 10 runs, in Figure 3.5. The network is initially attacked using the *RB* attack strategy (30 nodes removed), and the success of each defense is measured based on how it can reconnect the network by adding or rewiring edges in the network (higher is better). Based on Figure 3.5, we identify three key observations—(i) preferential edge addition performs the best; (ii) edge addition generally outperforms rewiring strategies; and (iii) random neighbor rewiring typically performs better than the other rewiring strategies.

3.4.3 Running Network Defenses in TIGER

The code block in Listing 3 illustrates how TIGER users can quickly setup a network defense simulation. There are 6 core parameters the user needs to set—the number of defense

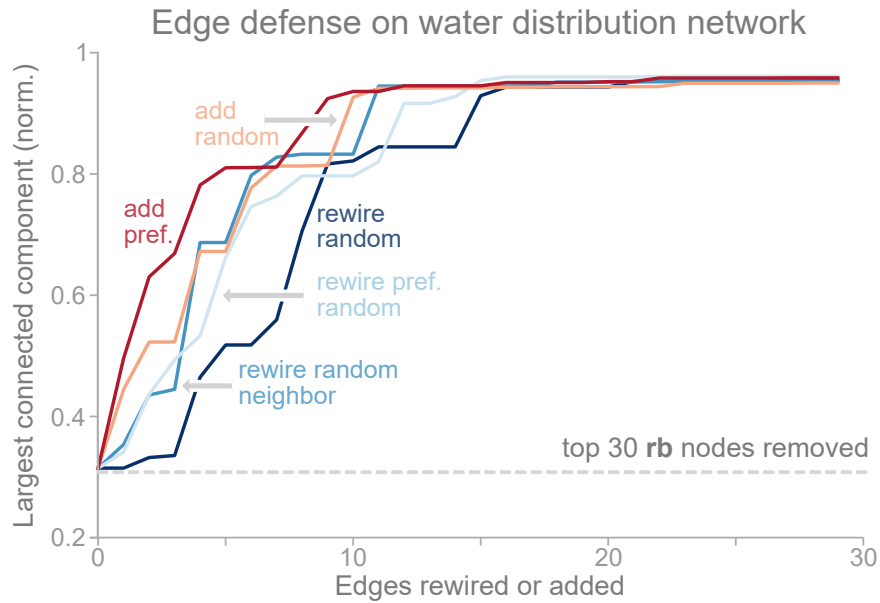


Figure 3.5: Comparing ability of 5 edge defenses to improve KY-2 network robustness after removing 30 nodes via RB attack. Edge addition performs the best, with random edge rewiring performing the worst.

simulations ‘runs’, the defense strategy ‘defense’, the number of edges to add or rewire ‘steps’, the attack strategy ‘attack’, and the number of nodes or edges to remove ‘k_a’. The output of the simulation is a plot showing the ability of the network to recover after it has been attacked.

```

1 from graph_tiger.defenses import Defense
2 from graph_tiger.graphs import graph_loader
3
4 params = {
5     'runs': 1,                # number of simulations
6     'steps': 30,              # rewire 1 edge per step
7     'defense': 'rewire_edge_preferential',
8     'attack': 'rd_node',      # attack strategy
9     'k_a': 30,                # attack strength
10 }
11
12 # Load Kentucky KY-2 water distribution graph
13 graph = graph_loader(graph_type='ky2')
14
15 # Run and plot defense simulation
16 d = Defense(graph, **params)
17 results = d.run_simulation()
18 d.plot_results(results)

```

3.5 TIGER Simulation Tools

We implement 4 broad and important types of robustness simulation tools [159, 102, 63, 23, 5]—(1) dissemination of network entities, (2) cascading failures (3) network attacks, see Section 3.3, and (4) network defense, see Section 3.4. In Section 3.5.3, we discuss the implementation of an infectious disease models and how defense techniques implemented in TIGER can be used to either *minimize* or *maximize* the network diffusion. Then, in Section 3.3, we discuss the implementation of the cascading failure model and its interactions with TIGER defense and attack strategies.

3.5.1 Cascading Failures

Cascading failures often arise as a result of natural failures or targeted attacks in a network. Consider an electrical grid where a central substation goes offline. In order to maintain the distribution of power, neighboring substations have to increase production in order to meet demand. However, if this is not possible, the neighboring substation fails, which in turn causes additional neighboring substations to fail. The end result is a series of cascading failures i.e., a blackout [52]. While cascading failures can occur in a variety of network types e.g., water, electrical, communication, we focus on the electrical grid. Below, we discuss the design and implementation of the cascading failure model and how TIGER can be used to both *induce* and *prevent* cascading failures using the attack and defense mechanisms discussed in Sections 3.3 and 3.4, respectively.

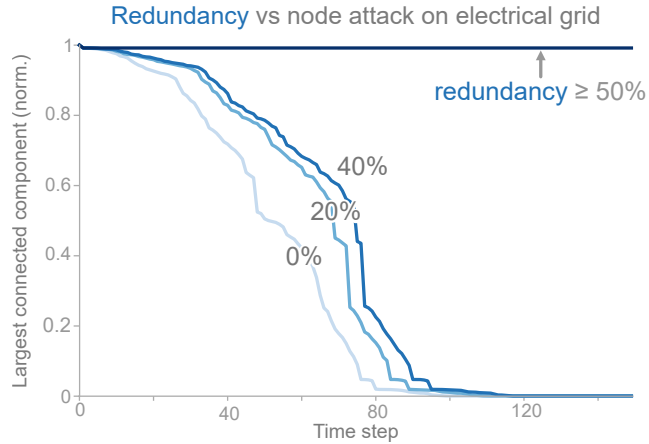


Figure 3.6: Effect of network redundancy r on the US power grid where 4 nodes are overloaded using ID. When $r \geq 50\%$ the network is able to redistribute the increased load.

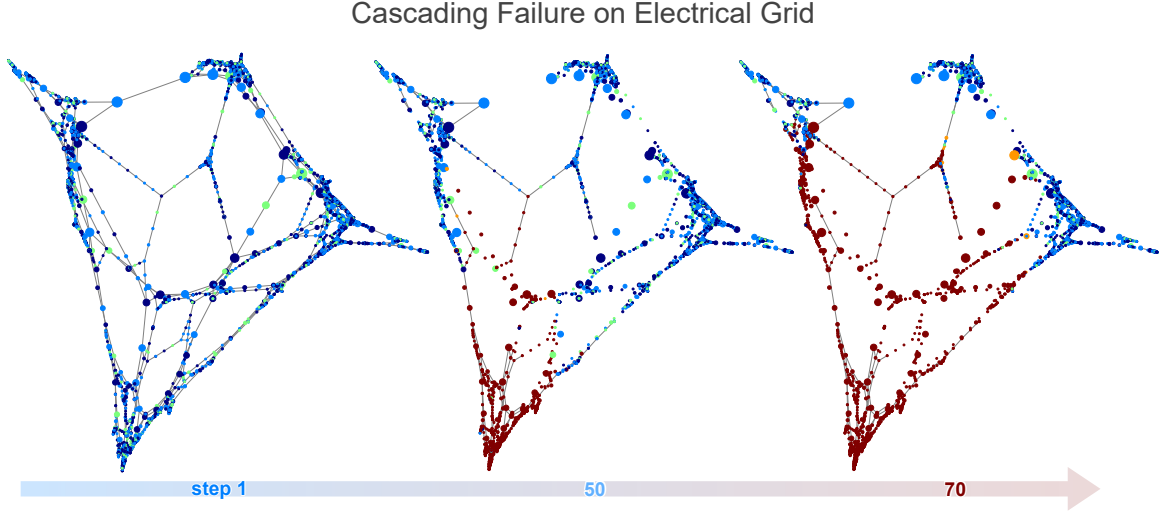


Figure 3.7: TIGER cascading failure simulation on the US power grid network when 4 nodes are overloaded according to the ID attack strategy. Time step 1: shows the network under normal conditions. Time step 50: we observe a series of failures originating from the bottom of the network. Time step 70: most of the network has collapsed.

Design and Implementation. There are 3 main processes governing the network simulation—(1) *capacity* of each node $c_v \in [0, 1]$; (2) *load* of each node $l_v \in U(0, l_{max})$; and (3) *network redundancy* $r \in [0, 1]$. The capacity of each node c_v is the the maximum load a node can handle, which is set based on the node’s normalized betweenness centrality [160]. The load of each node l_v represents the fraction of maximum capacity c_v that the node operates at. Load for each node c_v is set by uniformly drawing from $U(0, l_{max})$, where l_{max} is the maximum initial load. Network redundancy r represents the amount of reserve capacity present in the network i.e., auxiliary support systems. At the beginning of the simulation, we allow the user to attack and defend the network according to the node attack and defense strategies in Sections 3.3 and 3.4, respectively. When a node is attacked it becomes “overloaded”, causing it to fail and requiring the load be distributed to the neighbors. When defending a node we increase it’s capacity to protect against attacks.

Simulating cascading failures. To help users visualize cascading failures induced by targeted attacks, we enable them to create visuals like Figure 3.7, where we overload 4 nodes selected by the ID attack strategy on the US power grid dataset [102] ($l_{max} = 0.8$). Node size represents capacity i.e., larger size \rightarrow higher capacity, and color indicates the load of each node on a gradient scale from blue (low load) to red (high load); dark red indicates node failure (overloaded). Time step 1 shows the network under normal conditions; at step 50 we observe a series of failures originating from the bottom of the network; by step 70 most of the network has collapsed. To assist users in summarizing simulation results over many configurations, we enable them to create plots like Figure 3.6, which shows the effect

of network redundancy r when 4 nodes are overloaded by the ID attack strategy. At 50% redundancy, we observe a critical threshold where the network is able to redistribute the increased load. For $r < 50\%$, the cascading failure can be delayed but not prevented.

3.5.2 Running Cascading Failures in TIGER

The code block in Listing 4 shows how TIGER users can quickly setup a cascading failure simulation. There are 3 simulation specific parameters—the max node load ‘l’, node redundancy ‘r’, and maximum node capacity ‘c’ (based on betweenness centrality). We set the attack and defense parameters, similar to Listings 2 and 3, respectively. The simulation output is a plot measuring the ‘health’ or robustness of the network over time. Users can optionally generate image snapshots and a video simulation of the cascading failure on the network data.

```

1 from graph_tiger.cascading import Cascading
2 from graph_tiger.graphs import graph_loader
3
4 params = {
5     'runs': 1,                # number of simulations
6     'steps': 100,            # simulation time steps
7     'l': 0.8,                # max node load
8     'r': 0.2,                # node redundancy
9     'c': int(0.1 * len(graph)), # node capacity approx.
10
11     'robust_measure': 'largest_connected_component',
12     'k_a': 30,               # attack strength
13     'attack': 'rd_node',     # attack strategy
14     'k_d': 0,                # defense strength
15     'defense': None,         # defense strategy
16 }
17
18 # Load U.S. electrical grid graph
19 graph = graph_loader('electrical')
20
21 # Run and plot cascading failure simulation
22 cascading = Cascading(graph, **params)
23 results = cascading.run_simulation()
24 cascading.plot_results(results)

```

Listing 3.4: Cascading failure simulation on U.S. electrical grid using TIGER

3.5.3 Dissemination of Network Entities

A critical concept in entity dissemination is *network diffusion*, which attempts to capture the underlying mechanism enabling network propagation. In order to augment this diffusion process, TIGER leverages the defense techniques in Section 3.4 for use with two prominent diffusion models: the flu-like susceptible-infected-susceptible (SIS) model, and the vaccinated-like susceptible-infected-recovered (SIR) model [159]. For example, to *minimize* the ability of viruses to spread we can monitor (remove) nodes in the graph to reduce graph connectivity. On the other hand, if want to *maximize* network diffusion e.g., marketing campaign, we can use defense techniques like edge rewiring or addition to increase graph connectivity. Below, we highlight the SIS infectious disease model and how TIGER’s defense techniques can help contain a simulated outbreak.

Design and Implementation. Each node in the SIS model can be in one of two states, infected I or susceptible S . At each time step t , an infected node v has a probability β of infecting each of its uninfected neighbors $u \in N(v)$. After this, each infected node v has a probability δ of healing and becoming susceptible again. The relationship between the birth rate β , death rate δ and the spectral radius λ_1 of the graph has been a widely studied topic. In [124], they show that the spectral radius of a graph is closely tied to the epidemic threshold τ of a network in an SIS model. In particular, they prove that $\frac{\beta}{\delta} < \tau = \frac{1}{\lambda_1}$. This means for a given virus strength s , an epidemic is more likely to occur on a graph with larger λ_1 . As such, we say that a virus has an effective strength $s = \lambda_1 \cdot b/d$, where a larger s means a stronger virus [5].

Simulating dissemination of entities. To help users visualize the dissemination process, we enable them to create visuals like Figure 3.1, where we run an SIS computer virus simulation ($s = 3.21$) on the Oregon-1 Autonomous System network [4]. The top of Figure 3.1 shows the virus progression when defending 5 nodes selected by Netshield [5]. By time step 1000, the virus has nearly died out. The bottom of Figure 3.1 shows that the virus remains endemic without defense. To assist users in summarizing model results over many configurations, we enable them to create plots like Figure 3.8, which show results for 5 SIS effective virus strengths $s = \{0, 3.21, 6.42, 9.63, 12.84\}$ over a period of 5000 steps.

3.5.4 Running Entity Dissemination in TIGER

The code block in Listing 5 shows how TIGER users can run an entity dissemination simulation by setting a few key parameters—the type of entity simulation ‘model’ (e.g., SIS, SIR), the virus birth rate ‘b’, the virus death rate ‘d’, and the fraction of the network that starts off infected ‘c’. The simulation output is a plot of network infection over time. In

Virus Dissemination on AS Network

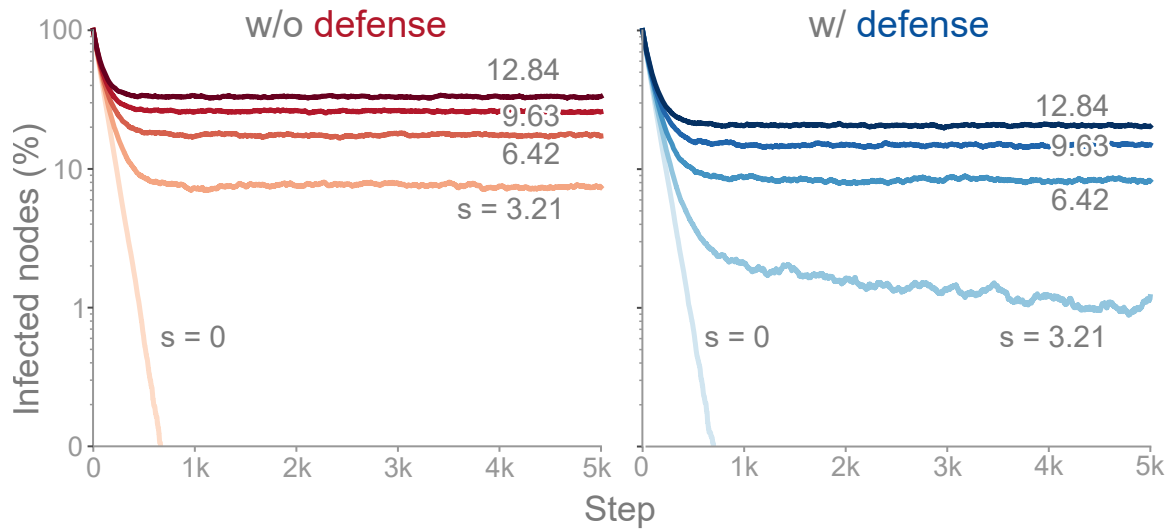


Figure 3.8: SIS simulation with 5 virus strengths on the Oregon-1 Autonomous System network. No defense (left), Netshield defense (right).

addition, users can optionally generate image snapshots and a video simulation of the entity dissemination on the network.

```

1 from graph_tiger.diffusion import Diffusion
2 from graph_tiger.graphs import graph_loader
3
4 sis_params = {
5     'runs': 1,          # number of simulations
6     'steps': 5000,      # simulation time steps
7
8     'model': 'SIS',
9     'b': 0.00208,      # virus birth rate
10    'd': 0.01,          # virus death rate
11    'c': 0.3,           # network % starting infected
12 }
13
14 # Load Oregon-1 Autonomous System graph
15 graph = graph_loader('as_733')
16
17 # Run and plot entity dissemination simulation
18 diffusion = Diffusion(graph, **sis_params)
19 results = diffusion.run_simulation()
20
21 diffusion.plot_results(results)

```

Listing 3.5: Entity dissemination simulation on Oregon-1 Autonomous System network using TIGER

3.6 Conclusion

The study of network robustness is a critical tool in the characterization and understanding of complex interconnected systems. Through analyzing and understanding the robustness of these networks we can: (1) quantify network vulnerability and robustness, (2) augment a network's structure to resist attacks and recover from failure, and (3) control the dissemination of entities on the network (e.g., viruses, propaganda). While significant research has been conducted on all of these tasks, no comprehensive open-source toolbox currently exists to assist researchers and practitioners in this important topic. This lack of available tools hinders reproducibility and examination of existing work, development of new research, and dissemination of new ideas. To address these challenges, we contribute TIGER, an open-sourced Python toolbox containing 22 graph robustness measures with both original and fast approximate versions; 17 failure and attack strategies; 15 heuristic and optimization based defense techniques; and 4 simulation tools. TIGER is open-sourced at: <https://github.com/safreital/TIGER>.

Part II

Robust Algorithms

Overview

While developing TIGER (Chapter 3), we found a dearth of graph measures that can effectively incorporate cybersecurity domain knowledge and measure the vulnerability of networks to lateral attacks. As such, we develop the D²M, the first framework that systematically quantifies network vulnerability to lateral attack and identifies at-risk devices from a graph theoretic perspective. Clicking on the link below will open its PDF version in the browser:

Chapter 4: D²M: Dynamic Defense and Modeling of Adversarial Movement in Networks Scott Freitas, Andrew Wicker, Duen Horng Chau, Joshua Neil. *SIAM International Conference on Data Mining (SDM)*. Online, 2020.
<https://arxiv.org/abs/2001.11108>

CHAPTER 4

D²M: DYNAMIC DEFENSE AND MODELING OF ADVERSARIAL MOVEMENT IN NETWORKS

Given a large enterprise network of devices and their authentication history (e.g., device logons), how can we quantify network vulnerability to lateral attack and identify at-risk devices? We systematically address these problems through D^2M , the first framework that models lateral attacks on enterprise networks using multiple attack strategies developed with researchers, engineers, and threat hunters in the Microsoft Defender Advanced Threat Protection group. These strategies integrate real-world adversarial actions (e.g., privilege escalation) to generate attack paths: a series of compromised machines. Leveraging these attack paths and a novel Monte-Carlo method, we formulate network vulnerability as a probabilistic function of the network topology, distribution of access credentials and initial penetration point. To identify machines at risk to lateral attack, we propose a suite of five fast graph mining techniques, including a novel technique called ANOMALYSHIELD inspired by node immunization research. Using three real-world authentication graphs from Microsoft and Los Alamos National Laboratory (up to 223,399 authentications), we report the first experimental results on network vulnerability to lateral attack, demonstrating D^2M 's unique potential to empower IT admins to develop robust user access credential policies.

4.1 Introduction

Attack campaigns from criminal organizations and nation state actors are quickly becoming one of the most powerful forms of disruption. In 2016 alone, malicious cyber activity cost the U.S. economy between \$57 and \$109 billion [2]. These cyber-attacks are often highly sophisticated, targeting governments and large-scale enterprises to interrupt critical services and steal intellectual property [3]. Unfortunately, once an attacker has compromised a single credential for an enterprise machine, the **whole network becomes vulnerable to lateral attack movements** [6], allowing the adversary to eventually gain control of the network (i.e., escalating privileges via credential stealing [7]).

Despite their prevalence, observing and analyzing lateral attacks is challenging for multiple reasons: (1) lateral attacks are still relatively sparse compared to the unsuccessful attack; (2) attack ground-truth is hard to ascertain, and generally partially uncovered through investigation; (3) incident reports are frequently withheld from the public for security and

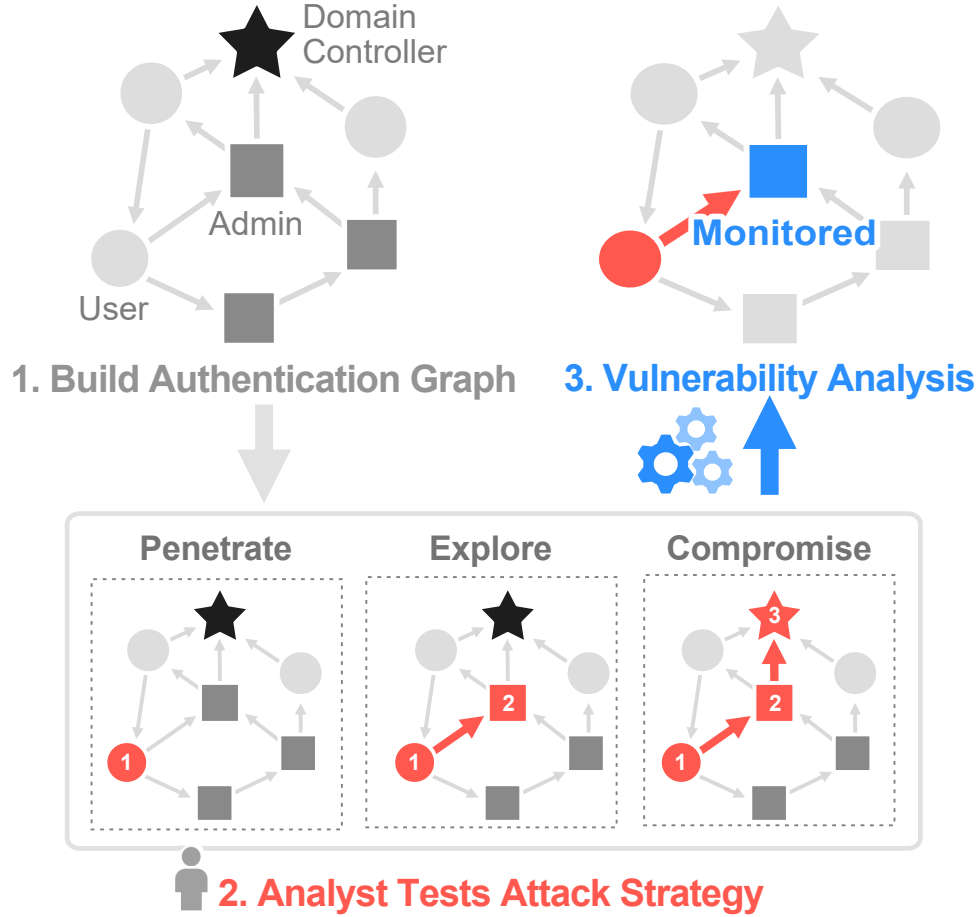


Figure 4.1: Our D^2M framework: **1.** Builds an authentication graph from device authentication history; **2.** Allows security analysts to test different attack strategies to study network vulnerability; **3.** Identifies at-risk machines to monitor, preempting lateral attacks.

privacy concerns; and (4) due to the fact that the adversary already has a valid credential for the network (e.g., gained through phishing [161]), attackers can operate as a legitimate user. While real attack data does exist—due to the above challenges, it is rarely fully visible, or accessible, making the study of a “complete” attack highly problematic.

Our Contributions We propose D^2M , the first framework that systematically quantifies network vulnerability to lateral attack and identifies at-risk devices (Fig. 4.1). Our major contributions include:

- **Attack Strategies** D^2M enables security researchers to integrate their crucial domain knowledge from studying prior attacks in the form of attack strategies. We developed three attack strategies by actively engaging researchers, engineers and threat hunters in the Microsoft Advanced Threat Protection group, whose expertise lies in tracking down adversaries in a post-breach environment (once adversary is on network). D^2M inte-

grates real-world adversarial actions (e.g., privilege escalation), generating attack paths consisting of a series of compromised machines (Sec. 4.5; Fig. 4.1.2).

- **Network Vulnerability Analysis** We formulate a novel Monte-Carlo method for lateral attack vulnerability as a probabilistic function of the network topology, distribution of access credentials and initial penetration point (Fig. 4.1.3). This empowers IT admins to develop robust user access credential policies and enables security researchers to study the vulnerability of a network to lateral attack (Sec. 4.6).
- **Network Defense by Identifying At-risk Machines** To identify machines at risk to lateral attack, we propose a suite of five fast graph mining techniques, including a novel technique called ANOMALYSHIELD which prioritizes machines with anomalous neighbors and high eigencentrality (Fig. 4.1.3; Sec. 4.7).
- **Evaluation Using Real-World Data** Using three real-world authentication graphs from Microsoft and Los Alamos National Laboratory (LANL; up to 223,399 authentications), we report the first experimental results on network vulnerability to lateral attack and at-risk machine identification (Sec. 4.5).
- **Impact to Microsoft and Beyond.** The Microsoft Defender Advanced Threat Protection product is deployed to thousands of enterprises around the world, and is a leader in the Endpoint Detection and Response (EDR) market [162]. The ability to detect and prevent lateral movement is one of the most challenging areas of post-breach detection. This research has led to major impact to Microsoft products, inspiring changes to the product’s approach to lateral movement detection.

Table 4.1 describes the main symbols used in the chapter. We follow standard notation and use capital bold letters for matrices (e.g., \mathbf{A}), lower-case bold letters for vectors (e.g., \mathbf{a}) and calligraphic font for sets (e.g., \mathcal{S}).

4.2 Background and Our Differences

Our work intersects the domains of lateral attack and graph mining, we briefly review related work below. Different from existing work that detects lateral movement after an adversary is on the network, our work **quantifies network vulnerability to lateral attack** and **identifies at-risk machines**. Another important distinction is that this work uses real-world enterprise authentication graphs, while most prior work has not.

Table 4.1: Symbols and Definition

Symbol	Definition
G	Directed, unweighted, attributed graph
\mathcal{V}, \mathcal{E}	Set of nodes and edges in graph G
n, m	Number nodes $ V $, edges in $ \mathcal{E} $ in G
$A(i, j)$	Adj. matrix of G at i th row, j th column
$u(i)$	Eigenvector at position i
\mathcal{C}, c	Credential set; credential instance
D	Credential generation process
d	Credential vector
\mathcal{H}, h	Ordered hygiene set; hygiene instance
$N^+(v), N(v)$	Successors of v ; neighbors of v
\mathcal{R}, \mathcal{T}	Set of start nodes; set of attacker moves
\mathcal{S}_k	Set of k nodes to monitor
$SV(\mathcal{S}_k)$	Shield value of \mathcal{S}_k
$AV(\mathcal{S}_k)$	Anomaly value of \mathcal{S}_k
$L(G)$	Vulnerability of G to lateral attacks
p	Attack path
a	Per-machine anomaly vector
i_s	Number of sub-path intervals
k	Number of machines to vaccinate

4.2.1 Detecting Lateral Attacks

Significant research in *detecting* lateral movement in networks has been done [163, 164, 165, 166]. Latte [163], a graph based detection framework, discovers potential lateral movement in a network using forensic analysis of known infected computers. In [164], Neil et al. detects lateral attacks using statistical detection of anomalous graph patterns (e.g., paths, stars) over time. Alternatively, Nouredine et al. [165] proposes a zero-sum game to identify which machines a defender should monitor to slow down an attacker. Finally, a data fusion technique is proposed by Fawaz et al. [166], where host-level process communication graphs are aggregated into system-wide communication graphs to detect lateral movement.

4.2.2 Graph Mining & Network Security

Graph mining has been extensively applied to the more general domain of network security. Authentication graphs have been used to study network security from a variety of viewpoints [6, 167, 164]. In [6], Hagberg et al. studies credential hopping in authentication graphs and finds that by reducing a machine’s credential cache, lateral movement can be restricted. Alternatively, Kent et al. [167] develops individual user authentication graphs

to differentiate normal authentication activity from malicious. Orthogonal to the authentication graph and *our work*, attack graphs have been proposed to analyze a network’s risk to known security issues [168, 169, 170]. These graphs represent sequences of known system vulnerabilities that can be maliciously exploited; and are often used by IT admins to determine patch priority.

4.3 Authentication Graph

D^2M converts authentication history of network devices into an *authentication graph*, where directed edges represent machine-machine authentications (i.e., logons) in an organization. Below, we provide an overview of the authentication graph setup and the infusion of real-world domain knowledge into its construction.

4.3.1 Building Graph Structure

Modern enterprise computer networks typically rely on one of two types of centrally managed authentication mechanisms to authenticate user activity: Microsoft NTLM [171] or MIT Kerberos [172]. To avoid repeated authentication with network resources (e.g., printer, corporate web sites, email), both NTLM and Kerberos implement credential caching where user credentials are stored on the computer until either the user logs off (Kerberos), or the machine is restarted (NTLM) [6]. While these cached credentials are convenient for legitimate user activity, they pose significant risk for malicious exploitation [7, 173].

Leveraging this authentication history, we form a directed, unweighted graph $G = (\mathcal{V}, \mathcal{E})$, where an edge represents an authentication between source machine v_s and destination machine v_d (see Fig. 4.1.1). We combine all authentications between two machines into a single edge. These authentication events are recorded over a period of time, forming the graph topology of an organization [6, 167]. To verify that a remote connection between two machines can be established, authentication information is passed using cached credentials. In an enterprise network, these credentials typically follow a hierarchical scheme: *user* (c_1) at the bottom, *local admin* (c_2) and *network admin* in the middle (c_3), and *domain admin* (c_4) at the top ($c_1 < c_2 < c_3 < c_4$) [173]. Depending on the type of cached credential, it will be valid until the user logs out (Kerberos) or until the machine is restarted (NTLM).

4.3.2 Integrating Domain Knowledge

To enhance D^2M with realistic security and attack practices, we integrate the following three components into our framework: (1) per-machine **credential caching**; (2) **network**

hygiene (i.e., how many ‘users’ and ‘admins’ on the network); and (3) **domain controller** modeling.

Credential Caching We embed attribute information into graph G by giving each machine $v \in V$ a cached credential. These credentials are stored as a vector $\mathbf{d} \in \mathbb{R}^n$, where each entry is a machine in the authentication graph containing the most recent credential $\mathbf{d}(i) = c$. While some credential schemes have additional levels and queue lengths as active directory policies, our approach captures representative security information.

Network Hygiene We model various credential distributions through three levels of hygiene $h \in \mathcal{H}$ due to the unavailability of credential information in the network $\mathbf{d} = \langle c_1, c_2, \dots, c_n \rangle$ where $n = |\mathcal{V}|$. Each hygiene level (h_1 : low, h_2 : medium, h_3 : high) represents the frequency with which credential types are observed on the network. Intuitively, a low hygiene level (h_1) models a network with loose IT policies and an abundance of high-level administrator credentials. In contrast, a high hygiene level (h_3) represents a network with strict IT policies and limited distribution of admin credentials. We select each hygiene distribution $h \in \mathcal{H}$ as: $h_1 = \{c_1: n, c_2: n/2, c_3: n/5, c_4: n/20\}$, $h_2 = \{c_1: n, c_2: n/4, c_3: n/10, c_4: n/50\}$ and $h_3 = \{c_1: n, c_2: n/8, c_3: n/20, c_4: n/80\}$, which are determined experimentally in conjunction with domain experts.

In practice, we distribute these credentials for a given hygiene h as follows. For every machine in the network $v \in \mathcal{V}$ we assign the lowest authorization level $\mathbf{d}(v) = c_1$. We then distribute higher level credentials as follows—for each increasing credential level $c \in \{c_2, c_3, c_4\}$, we randomly select $h(c)$ machines from \mathcal{V} and loop through each one, replacing its credential level with a higher one. While these distributions cannot match every organization’s IT policies, we select them to model a broad range.

Domain Controller & Privilege Escalation The final component we model is the domain controller, which controls access to network resources. When a source machine v_s attempts to establish a remote connection to a destination machine v_d , the domain controller determines if v_s has sufficient privileges $\mathbf{d}(v_s) \geq \mathbf{d}(v_d)$. Since an organization’s domain controller(s) are never observed with certainty, we identify it using PageRank ($\alpha=0.15$) [140]—assigning the machine with largest PageRank vector $\mathbf{r} \in \mathbb{R}^n$ component the role of domain controller $v_{dc} = \text{argmax}(\mathbf{r})$. After discussions with domain experts, we make the simplifying assumption that the machine with largest PageRank is the domain controller v_{dc} , since it often has the largest number of incoming edges (from incoming authentication requests).

Finally, we incorporate the concept of privilege escalation by allowing the attacker to connect to a machine that is one credential level higher. That is, if the attacker has collected

credentials c_1 and c_2 , they can connect to a c_1 , c_2 , or c_3 machine. In practice, this is done through mining the memory of the machine to gain higher levels of credential [174].

4.4 Formulating the Research Problems

We formally define the three problems that D^2M addresses below. Then we present our solutions for them in **Section 4.5, 4.6, and 4.7** respectively.

Problem 1 *Lateral Attack Modeling*

Given: *an attack strategy, an initial penetration point, and directed unweighted graph G with associated credential distribution $\mathbf{d} \in D$*

Find: *an attack path $\mathbf{p} = \langle v_1, v_2, \dots, v_i, \dots, v_t \rangle$ in graph G that starts from the penetration point and reaches the domain controller, while escalating privileges in increasing order (see Fig. 4.2)*

Problem 2 *Lateral Attack Vulnerability*

Given: *graph G with credential distribution $\mathbf{d} \in D$*

Measure: *vulnerability $L(G)$ to lateral attacks*

Problem 3 *Lateral Attack Defense*

Given: *graph G with credential distribution $\mathbf{d} \in D$, and suspected adversary movement p*

Identify: *k best machines to monitor for attacks*

4.5 D^2M : Lateral Attack Modeling

We present our solution for the *lateral attack modeling* problem (Sect. 4.4: Problem 1). We begin with an overview of the lateral attack process in Section 4.5.1. Section 4.5.2 presents lateral attack strategies—developed with Microsoft domain experts—that produce lateral movement. Section 4.5.3 details the algorithm for modeling lateral attacks on authentication graphs.

4.5.1 Lateral Attack Overview

An enterprise attack typically follows a kill chain, which can be distilled into three phases—(1) penetration of the network; (2) exploration of the network and escalation of privileges; and (3) exfiltration of data back to the command and control server [175]. We discuss each phase below and highlight our modeling assumptions.

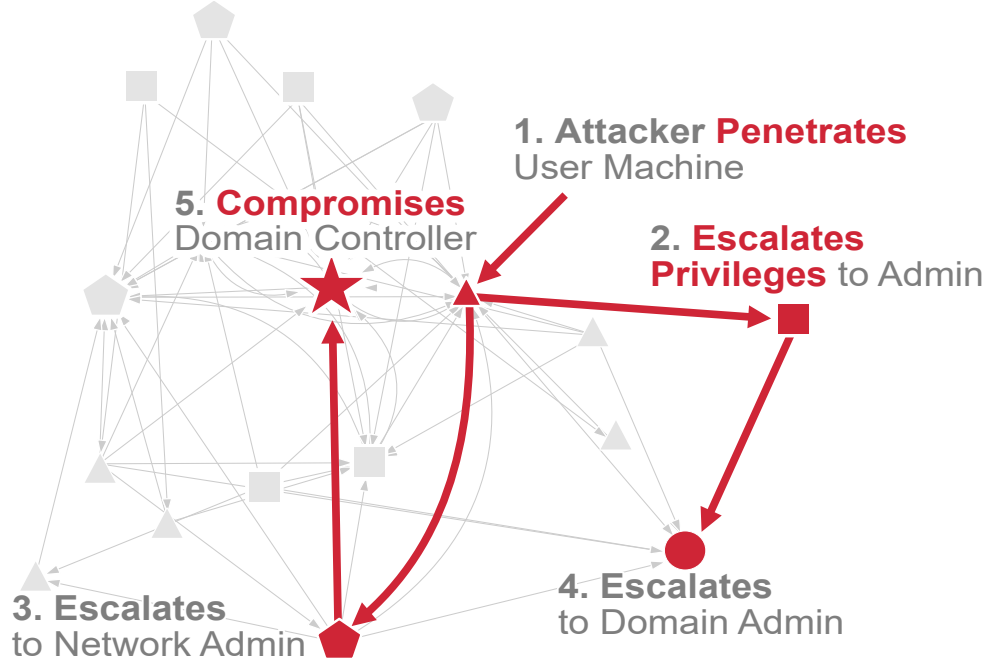


Figure 4.2: Attack path generated by D^2M . **1.** Network is penetrated; **2-4.** Attacker explores the network and escalates privileges; **5.** Attacker compromises the domain controller, gaining control of the network.

Penetration An enterprise network is typically penetrated through two mechanisms—(a) phishing campaigns targeting organization employees or (b) incidental exposure from employees downloading malware on high-risk websites (drive-by download) [176]. We assume the former, since sophisticated adversaries often target enterprise networks for penetration. A phishing campaign begins by targeting organization employees through authentic looking emails containing malicious attachments or web links. These malicious attachments contain malware that installs a backdoor; once a backdoor is installed the attacker gains remote access to the machine, penetrating the enterprise network. We model this penetration process by assuming that most compromised employees (machines) $v \in \mathcal{V}$ are at the c_1 credential level and let the attacker randomly start on any of these machines $\mathcal{R} = \{v \in \mathcal{V} \mid d(v) = c_1\}$.

Explore & Exploit Once an adversary is on a network, their goal is to explore the network and escalate privileges. This process begins by stealing the infected machines cached credentials, allowing them to authenticate with neighboring machines. These credentials can be stolen in a number of ways, however, it is beyond the scope of this work and we refer the reader to [173]. Once the adversary has connected to a neighboring machine, they again steal the cached credentials [7] and continue this process until they have obtained domain

admin privileges c_4 . We model this attack process in two ways—(1) black-box, where the attacker has no prior information on the network (i.e., normal pattern of authentications); and (2) gray-box, where the attacker has prior information on the network layout, possibly through prior reconnaissance or inside help.

Exfiltrate After the adversary has obtained a domain admin credential c_4 , they are able to connect to any networked machine, freely exploring the network until they reach the domain controller. Upon accessing the domain controller, the attacker gains full control over the network. At this point the adversary can sweep the network for valuable information and exfiltrate with impunity. We leave modeling this aspect of the kill chain to future work.

4.5.2 Lateral Attack Strategies

In conjunction with domain experts, we develop three attack strategies to model lateral attacks on authentication graphs; one black-box and two gray-box.

Black-Box Attack

In the black-box setting we assume the attacker has no knowledge about the network and model movement through a modified random walk called *RandomWalk-Explore* (RWE).

RandomWalk-Explore (RWE) with 0.85 probability draws a machine v uniformly at random from the set of unvisited neighboring machines \mathcal{T} . With probability 0.15, the attacker randomly jumps with uniform probability to a machine in \mathcal{R} ; this helps to model some of the usual behavior that can occur during an attack (e.g., when an attacker finds remote machine information in plain-text). In addition, we select 0.15 as the random jump probability to align with information retrieval literature [140]. We model the RWE process in Equation 4.1, which describes the probability mass function (PMF) of a discrete random variable X_1 , which can take on any value in the range $R_{X_1} = \mathcal{T} \cup \mathcal{R}$ with probability $P_{X_1}(v)$. Based on discussions with threat hunters, we believe this form of uninformed exploration is the most realistic of the attack strategies.

$$P_{X_1}(v) = \begin{cases} 0.15/|\mathcal{R}|, & \text{if } v \in \mathcal{R} \\ 0.85/|\mathcal{T}|, & \text{if } v \in \mathcal{T} \\ 0, & \text{otherwise} \end{cases} \quad (4.1)$$

Gray-Box Attacks

In the gray-box setting, the attacker has additional information in the form of the network topology—allowing for informed attack strategies. We propose two strategies, *Rank-*

Explore (RE) and *Degree-Explore* (DE).

Rank-Explore (RE) with 0.85 probability draws a machine v at random from the set of unvisited neighboring machines \mathcal{T} with weight proportional to its PageRank vector r . With probability 0.15, the attacker randomly jumps with uniform probability to a machine in \mathcal{R} . This process is modeled in Equation 4.2.

$$P_{X_2}(v) = \begin{cases} 0.15/|\mathcal{R}|, & \text{if } v \in \mathcal{R} \\ 0.85 \cdot r(v) / \sum_{i \in \mathcal{T}} r(i), & \text{if } v \in \mathcal{T} \\ 0, & \text{otherwise} \end{cases} \quad (4.2)$$

Degree-Explore (DE) with 0.85 probability draws a machine $v \in \mathcal{T}$ with weight proportional to the distribution of the network's degree vector $\delta = \text{diag}(\mathbf{A} \cdot \mathbf{e})$. With probability 0.15, the attacker randomly jumps with uniform probability to a machine in \mathcal{R} . This process is modeled in Equation 4.3.

$$P_{X_3}(v) = \begin{cases} 0.15/|\mathcal{R}|, & \text{if } v \in \mathcal{R} \\ 0.85 \cdot \delta(v) / \sum_{i \in \mathcal{T}} \delta(i), & \text{if } v \in \mathcal{T} \\ 0, & \text{otherwise} \end{cases} \quad (4.3)$$

After a neighbor v has been selected by the attack strategy, we check that the attacker has the required credential level to visit this machine. For example, if c_2 is the current highest collected credential, then the attacker can move to any machine with credential level c_1 , c_2 , or c_3 . If the move is valid, we update the set of unvisited neighbors \mathcal{T} according to Equation 4.4 and allow the attacker to collect that machine's credential.

$$\mathcal{T} = \mathcal{T} \setminus \{v\} \cup N^+(v) \quad (4.4)$$

4.5.3 Lateral Attack Algorithm

We allow the attacker to randomly penetrate various points of the network ($v \in R$) and then move according to one of the three strategies: RWE, RE and DE, until the domain controller v_{dc} is reached or there are no neighbors to visit. Each successful run of this simulation generates an attack path $\mathbf{p} = \langle v_1, v_2, \dots, v_i, \dots, v_{dc} \rangle$, representing the sequence of machines visited, with the last node v_{dc} representing the domain controller. This process is modeled in Algorithm 1 and repeated for multiple credential distributions $\mathbf{d} \in D$ to eliminate bias from a single distribution. An example attack path generated from Algorithm 1 can be seen in Figure 4.2.

Algorithm 1: Lateral Attack Modeling

Input: Adj. matrix \mathbf{A} , h , attack strategy
Result: Attack pattern p

```
1 let  $r_o = \text{PageRank}(\mathbf{A})$ ; and  $\delta_o = \text{diag}(\mathbf{A} \cdot \mathbf{1})$ 
2 let  $d \sim D_h$  // distribute credentials
3  $\mathcal{R} = \{v \in V \mid d(v) = c_1\}$  // start nodes
4  $v = \text{rand}(\mathcal{R})$ ; let  $\mathcal{T} = N^+(v)$ ;  $p = [v]$ 
5  $\text{tried} = \{\}$ ;  $\text{visited} = \{\}$ 
6 while  $v \neq v_{dt}$  and  $|\mathcal{T}| > 0$  and  $|\text{tried}| < |\mathcal{T}|$  do
7    $\mathcal{T} = \mathcal{T} \setminus \text{tried}$ 
8   if  $\text{attack\_strategy} == RWE$  then
9      $v \leftarrow X_1$ 
10  else if  $\text{attack\_strategy} == RE$  then
11     $r = r_o(\mathcal{T})$ ;  $v \leftarrow X_2$ 
12  else if  $\text{attack\_strategy} == DE$  then
13     $r = \delta_o(\mathcal{T})$ ;  $v \leftarrow X_3$ 
14   $\mathcal{T} = \mathcal{T} \cup \text{tried}$ 
15  if  $\text{Valid}(v)$  and  $v \notin \text{visited}$  then
16     $\text{tried} = \{\}$ 
17     $\mathcal{T} = \mathcal{T} \setminus \{v\} \cup N^+(v)$ 
18     $p \leftarrow p \cup v$ ;  $\text{visited} \leftarrow \text{visited} \cup v$ 
19  else
20     $\text{tried} \leftarrow \text{tried} \cup v$ 
21 Return  $p$ 
```

4.5.4 Analysis of Lateral Attack Algorithm

The time and space complexity of Algorithm 1 is $O(n^2)$ and $O(n + m)$, respectively.

There are two time expensive computations, PageRank $O(n)$; and attack strategy machine selection inside the while loop $O(n)$. Since the while loop can visit every node in the graph, the worst case complexity will be $O(n^2)$. Space is linear with respect to nodes and edges $O(n + m)$ in the graph. Detailed proofs are omitted to save space.

4.6 D²M: Lateral Attack Vulnerability

We present our solution for the *lateral attack vulnerability* problem (Sect. 4.4: Problem 2). We begin by discussing the importance of network vulnerability scoring. We then formally introduce our method of measuring a network's vulnerability to lateral attacks. Finally, we discuss alternative graph vulnerability scores and why they are less suited to the task of measuring vulnerability to lateral movement.

Vulnerability Scoring To make data driven decisions regarding IT policy in an enterprise network, it is important to quantify the risk a network faces to lateral movement. Unfortunately, directly measuring this risk is difficult, requiring complex interactions of many unknown variables. To simplify these interactions, we propose to quantify network vulner-

ability to lateral attack $L(\cdot)$ as a function of three random variables—(1) network topology G , (2) distribution of credentials $\mathbf{d} \in D$ and (3) initial point of penetration $v \in \mathcal{R}$.

Since the true credential distribution $\mathbf{d} = \langle c_1, c_2, c_3, c_4 \rangle$ is unknown, along with knowledge of the organizations IT policies (strict, loose: Section 4.3.1), we model credential distributions through the use of hygiene levels $h \in \mathcal{H}$. For a given hygiene level $h \in \mathcal{H}$, we can marginalize out the dependency of the vulnerability score to the credential distribution $\mathbf{d} \in D_h$ in expectation, reducing the vulnerability score to $L(G, \mathcal{H} = h, V = v)$. In addition, we can simulate the attacker penetrating many different points in the network $v \in \mathcal{R}$, allowing us to marginalize out the dependency to v and reduce the score to $L(G, h)$. We can view this process in Equation 4.5 through the lens of Monte Carlo simulation, where in expectation we compute the graph vulnerability across many different credential distributions $\mathbf{d} \in D$ and start nodes $v \in \mathcal{R}$.

$$L(G, h) = \frac{1}{|D_h|} \frac{1}{|\mathcal{R}|} \sum_{\mathbf{d} \in D_h} \sum_{v \in \mathcal{R}} f(G, \mathbf{d}, v) \quad (4.5)$$

The vulnerability score $L(G, h)$ is a real number between $0 \leq L(G, h) \leq 1$, where a higher value indicates a more vulnerable network for the given topology G and hygiene level h . Intuitively, this score is saying that a network is more vulnerable if attacks are on average more successful for many credential distributions $\mathbf{d} \in D$ and penetration points $v \in \mathcal{R}$. We measure an attack's success through $f(\cdot)$, which simulates an attack using Algorithm 1. A value of $f(G, \mathbf{d}, v) = 1$ indicates a successful attack, which we define as being able to reach the domain controller v_{dc} . Future work could generalize this to other targets such as high value servers.

We further simplify the vulnerability score $L(\cdot)$ by marginalizing out the dependency to hygiene level $h \in \mathcal{H}$. This simplifies Equation 4.5 to a function of the network topology G , as seen in Equation 4.6.

$$L(G) = \sum_{h_i \in \mathcal{H}} p(h_i) \cdot L(G, h_i) \quad (4.6)$$

With no prior knowledge on the true distribution of hygiene levels in an organization, we assume a uniform prior $p(h) = 1/3$.

Alternative Scoring Significant work has gone into measuring the vulnerability of graphs [49, 89, 5]. For example, in [5] the authors define vulnerability of an undirected graph G as the largest eigenvalue $L(G) \triangleq \lambda$ of the adjacency matrix. The intuition is that as the largest eigenvalue increases, so does the path capacity of the graph. However, this form of topological vulnerability scoring can only indirectly measure the vulnerability of the graph to lateral movement since no security domain knowledge is integrated.

4.7 D²M: Lateral Attack Defense

We present our solution for the *lateral attack defense* problem (Sec. 4.4: Problem 3), where the objective is to identify the best set of k machines \mathcal{S}_k to monitor for lateral attacks. Once this set of machines \mathcal{S}_k has been identified, multiple safeguards can be implemented, including: changing the sensitivity of on device machine learning models and force resetting the password.

We make the following assumptions during the defense process—(a) there exists per-machine anomaly detection models that alert on unusual behavior (e.g., deviation in port or process activity). Since behavioral deviations have a larger false positive rate, their behavior is anomalous but not necessarily malicious. For this reason, anomaly alerts are ill-suited for investigation in isolation due to low confidence. However, these deviation scores are useful for machine monitoring decisions, especially when these alerts aggregate together [164]. (b) We assume that each anomaly detection model is providing real-time feedback to the defender; and (c) that the defender views all anomalous activity as it occurs through the system alerts. While assumption (c) is strong, we leave it to future work to model partial information defense strategies.

4.7.1 Defense Strategies

We propose a suite of five defense strategies, three *static* and two *dynamic*. A *static* strategy takes into account only the network topology G ; useful for protecting machines when monitoring resources are limited. A *dynamic* strategy considers both the network topology G and suspected lateral path movement activity $\mathbf{p}^t, \mathbf{p}^{t-1}, \dots, \mathbf{p}^i, \dots, \mathbf{p}^0$, where $\mathbf{p}^i \in \mathbb{R}^n$ is a sub-path containing suspicious activity in a given interval. This could be useful for real-time protection malicious activity investigation.

Each attack path \mathbf{p} is divided into i_s sub-paths, where each sub-path \mathbf{p}^i is of equal size (except for, possibly, the last sub-path \mathbf{p}^t) where $t \in [0, \lceil \frac{\mathbf{p}}{i_s} \rceil]$. A larger value of i_s creates a few long sub-paths, which could represent fast moving attacks in the network; conversely, a small i_s creates many short sub-paths, representing slow attacks.

Rank-Defense (RD) statically identifies at-risk machines based on the network’s PageRank [140]. Assuming a sorted PageRank vector, we identify machines as follows: $\mathcal{S}_k = \cup_{i=1}^k \mathbf{r}_i$.

Degree-Defense (DD) statically vaccinates the network according to the machines in the network with highest degree. With a sorted degree vector, we identify machines as follows: $\mathcal{S}_k = \cup_{i=1}^k \delta_i$. While RD and DD are simple defensive strategies, we are not aware of any work proposing to identify at-risk machines to lateral attacks using them.

NetShield (NS) [5] statically vaccinates the network according to the machine's Shield-Value (SV) in Equation 4.7. The actual selection of \mathcal{S}_k occurs in conjunction with the NetShield algorithm from [5], where the intuition is to select nodes with highest eigencentrality [177] while enforcing distance between selected machines (small or zero $\mathbf{A}(i, j)$). Here, $\mathbf{A} \in \{0, 1\}^{n \times n}$, λ is the largest eigenvalue, and \mathbf{u} is the associated eigenvector.

$$SV(\mathcal{S}_k) = \sum_{i \in \mathcal{S}_k} 2\lambda \cdot \mathbf{u}(i)^2 - \sum_{i, j \in \mathcal{S}_k} \mathbf{A}(i, j) \mathbf{u}(i) \mathbf{u}(j) \quad (4.7)$$

Random Anomalous Neighbor Defense (RAND) dynamically identifies machines by selecting an anomalous machine v_a with weight proportional to its anomaly score $\mathbf{a}(v_a)$, where each element $\mathbf{a}(v) \in [0, 1]$ and $\mathbf{a} \in \mathbb{R}^n$. We assume that when an alert is generated for a machine in a sub-path, it produces a value of $\mathbf{a}(v) = 1$, repeating for every machine $v \in \mathbf{p}^i$. After machine monitoring set \mathcal{S}_k is identified using sub-paths $\mathbf{p}^i, \dots, \mathbf{p}^0$, the anomaly scores are decayed $\mathbf{a}^{t+1} = \mathbf{a}^t/2$ to give weight to recent activity (determined experimentally).

The RAND strategy is described through Equations 4.8 and 4.9. Eq. 4.8 describes the PMF of discrete random variable X_4 , which can take on any value in the range $R_{X_4} = \{v \in \mathcal{V} \mid \mathbf{a}(v) > 0\}$ with probability $P_{X_4}(v)$. After drawing a machine $v_a \sim X_4$, we uniformly at random select a neighbor from v_a . This can be seen in Equation 4.9, which describes the PMF of discrete random variable X_5 , where X_5 can take on any value in the range $R_{X_5} = N^+(v_a)$ with probability $P_{X_5}(v)$. This process repeats until k machines have been selected.

$$P_{X_4}(v) = \begin{cases} \mathbf{a}(v) / \sum_{i \in \mathcal{V}} \mathbf{a}(i), & \text{if } v \in R_{X_4} \\ 0, & \text{otherwise} \end{cases} \quad (4.8)$$

$$P_{X_5}(v) = \begin{cases} 1/|N^+(v_a)|, & \text{if } v \in N^+(v_a) \\ 0, & \text{otherwise} \end{cases} \quad (4.9)$$

ANOMALYSHIELD (AS), a novel method we introduce for dynamic machine identification. We select machines for monitoring according to their ANOMALYVALUE (AV) in Equation 4.10, in combination with ANOMALYSHIELD (Algorithm 2). The intuition is that we prioritize machines with anomalous neighbors and high eigencentrality.

$$AV(\mathcal{S}_k) = \sum_{i \in \mathcal{S}_k} \mathbf{u}(i) \sum_{j \in N(i)} \mathbf{a}(j) \mathbf{u}(j) \quad (4.10)$$

Since both NetShield and AnomalyShield use eigenvector centrality as the underlying centrality metric, we convert the directed authentication graphs to undirected ones for use in the strategies.

Algorithm 2: ANOMALYSHIELD

Input: Adjacency matrix A , anomaly vector a , and vaccination budget k
Result: a set \mathcal{S}_k with k nodes

- 1 Compute first eigenvalue λ and corresponding eigenvector u of A
- 2 $c = A * (a * u)$
- 3 $score = c * u$
- 4 **for** $iter = 1$ to k **do**
- 5 $v = \operatorname{argmax}_i score(i)$, add v to set \mathcal{S}
- 6 $score(v) = -1$
- 7 **return** \mathcal{S}

4.7.2 Analysis of Defense Strategies

We evaluate time and space complexity with respect to each strategy since they are the dominating defense cost. The space is uniform across strategy $O(n + m + k)$, with time complexity shown below.

$$Time = \begin{cases} O(n \log n), & \text{if defense = RD} \\ O(n \log n), & \text{if defense = DD} \\ O(nk^2 + m), & \text{if defense = NS [49]} \\ O(kn + m), & \text{if defense = AS} \\ O(kn), & \text{if defense = RAND} \end{cases} \quad (4.11)$$

4.8 Experiments

4.8.1 Experimental Setup

All experiments are conducted on three real authentication graphs, collected over 30 days (statistics in Table 4.2). Two graphs are from Microsoft: anonymized enterprise networks G_s and G_l ; and one is from Los Alamos National Lab [178]: open-sourced network G_{lanl} . For each attack strategy and hygiene level, we strive to collect 200 unique attack paths for 50 credential distributions $d \in D$. These parameters are determined based on the available 2-week computation budget for data collection. Certain combinations of G and d have a high rate of attack failure; we terminate the collection process at 10,000 failed attempts, collecting as many as possible.

Table 4.2: Graph Statistics. ρ : graph density, C : average clustering coefficient, δ_{avg} : mean node out-degree.

Graph	Source	$ V $	$ E $	ρ	C	δ_{avg}
G_s	Microsoft	100	279	0.028	0.23	5.58
G_l	Microsoft	2,039	3,853	0.001	0.26	3.78
G_{lanl}	LANL	14,813	223,399	0.001	0.62	30.16

4.8.2 Network Vulnerability Analysis

In Table 4.3, we summarize the first experimental results on network vulnerability to lateral attack by analyzing the attack strategies *Rank-Explore* (RE), *Degree-Explore* (DE), and *RandomWalk-Explore* (RWE) (discussed in Sect. 4.5). For each strategy, we average the attack path length across all credential distributions. We compute the network vulnerability statistics using Eq. 4.5—hygiene-specific $L(G, h)$; and Eq. 4.6—whole-network $L(G)$ from Section 4.6. We identify multiple key insights:

1. **Informed Strategies Lead to Quicker Attacks** The RE and DE strategies produce shorter paths in general, compared to RWE. This is expected, as prior knowledge should help the attacker reach the domain controller in less time. Also, adversaries likely prefer shorter attack paths, which leaves smaller footprints for anomaly systems to detect.
2. **Improving Hygiene Reduces Vulnerability** Increasing network hygiene ($h_1 \rightarrow h_2 \rightarrow h_3$) causes longer attack paths (or none at all) and generally reduces vulnerability (e.g., for G_s and G_l). On graph G_s , the highest hygiene level h_3 critically reduces high-level admin credentials, significantly improving network robustness (vulnerability reduced to 0). Such findings can empower IT admins to develop robust user access credential policies.
3. **Linking Topology to Network Vulnerability** Networks that are well-connected are more vulnerable to lateral attack (e.g., G_{lanl} , with higher average clustering coefficient and node degree). This is expected, due to increased lateral movement opportunities. Relatedly, improving network hygiene level in such a well-connected network does not seem to reduce network vulnerability.

4.8.3 Defense Strategy Analysis

We report the first results for identifying machines at-risk to lateral attack, evaluating each defense strategy proposed in Section 4.7. We measure the success of each strategy by its ability to predict attacker movement. That is, given graph topology G and suspected lateral attack movement p^i, \dots, p^0 , predict attack activity at p^{i+1} (each p^i is a sequence/path of

Table 4.3: Vulnerability Statistics. Statistics excluded for G_{lanl} strategies RE and DE in h_3 as computation exceeded budget (Sect. 4.8.1).

Graph	Hygiene	Avg. Path length			Vulnerability	
		RE	DE	RAND	$L(G, h)$	$L(G)$
G_s	h_1	19	19	25	.773	
	h_2	49	39	39	.801	.525
	h_3	0	0	0	0	
G_l	h_1	33	36	46	.005	
	h_2	63	63	68	.006	.005
	h_3	133	139	139	.004	
G_{lanl}	h_1	22	18	45	.967	
	h_2	88	128	90	.981	.976
	h_3	-	-	249	.981	

suspected machines traversed by the attacker). Formally, we intersect the predicted *at-risk machines* \mathcal{S}_k with \mathbf{p}^{i+1} . Since the defender likely monitors the domain controller, we exclude it from \mathcal{S}_k . We repeat this process for each sub-path (except \mathbf{p}^0) and average over all attack paths. Figure 4.3 shows every combination of attack and defense strategy, with budget $k=8$ and hygiene h_2 , which provide representative results. We identify multiple key insights:

1. **ANOMALYSHIELD as Effective General Defense** **ANOMALYSHIELD** generally performs well (identified more machines) across: network topology (rows in figure), adversary’s prior knowledge (columns), and attack speed (horizontal axes). We believe this is because ANOMALYSHIELD focuses on high-centrality machines with anomalous neighbors, combining desirable attributes from static and dynamic methods.
2. **Similar Effectiveness in Small Graphs** All strategies perform similarly in small graph G_s (first row), since fewer machines exist for monitoring.
3. **Large Graphs Require Informed Defense** Uninformed defense strategy **RAND** is significantly less effective in the large graph G_{lanl} (last row), especially when encountering faster attacks. This could be explained by the need for intelligent decision making in the presence of many options.

4.9 Conclusion

We present D^2M , the first framework that systematically quantifies network vulnerability to lateral attacks and identifies at-risk devices. D^2M models lateral attacks on enterprise networks using attack strategies developed with Microsoft. We formulate network vul-

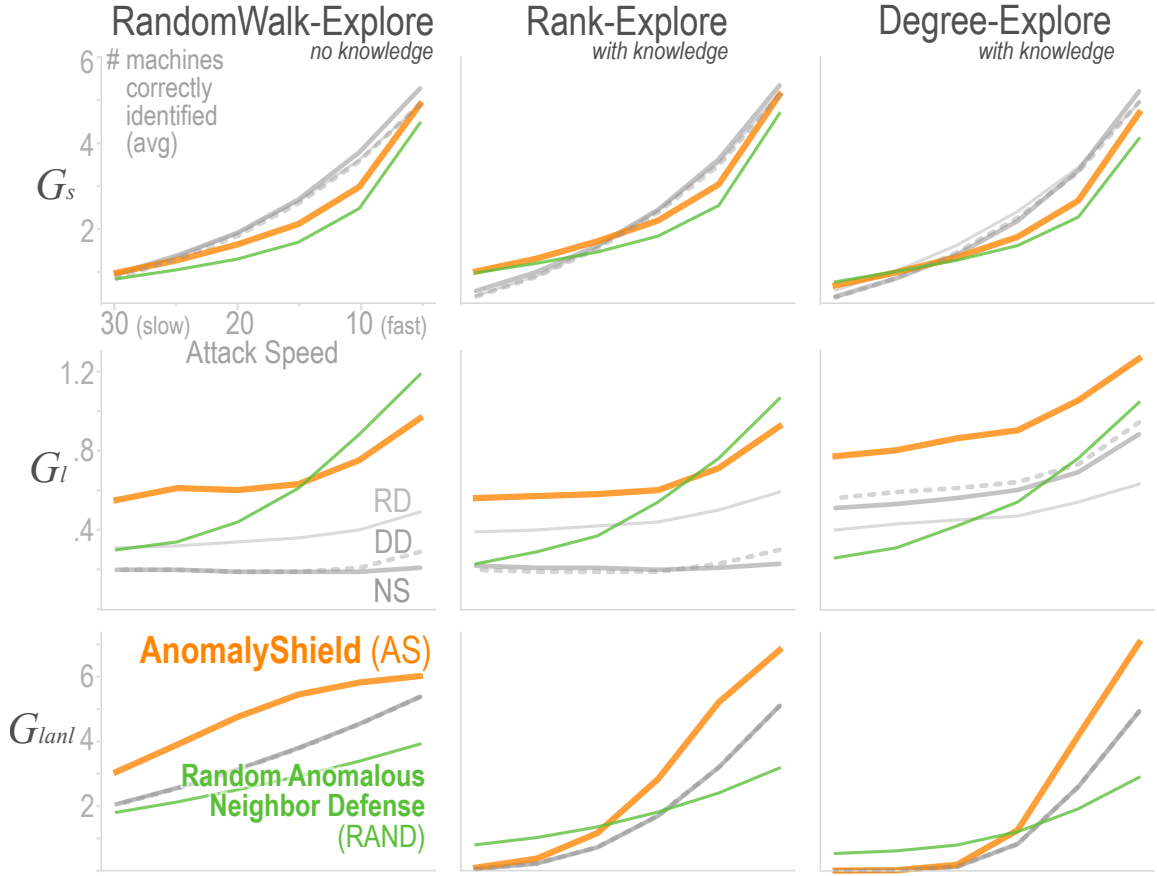


Figure 4.3: Each defense strategy is compared on three graphs and attack strategies, where ANOMALYSHIELD performs well across a majority of application scenarios.

nerability as a novel Monte-Carlo method and propose a suite of five fast graph mining techniques, including the novel ANOMALYSHIELD method, to identify at-risk machines. Using real data, we demonstrate D^2M 's unique potential to empower IT admins to develop robust user access credential policies.

Part III

Robust Databases

Overview

To prevent lateral attacks altogether (Chapter 4), we developed MALNET-GRAPH the world’s largest cybersecurity graph database—containing over 1.2 million graphs across 696 classes—and show the first large-scale results demonstrating the effectiveness of malware detection through a graph medium. Clicking on the link below will open its PDF version in the browser:

Chapter 5: Large-Scale Database for Graph Representation Learning

Scott Freitas, Yuxiao Dong, Joshua Neil, Duen Horng Chau. *Neural Information Processing Systems (NIPS) Datasets and Benchmarks, 2021*. <https://arxiv.org/abs/2011.07682>

Following up on our ground-breaking database MALNET-GRAPH, we extend the modality of data to incorporate image data through the formation of binary-images which represent the bytecode of malicious software (MALNET-IMAGE). To date, MALNET-IMAGE is the largest publicly available cybersecurity image database, offering $24\times$ more images and $70\times$ more classes than the only other public binary-image database. In total, MALNET-IMAGE will contain over 1.2 million images across a hierarchy of 47 malware types and 696 malware families. Clicking on the link below will open its PDF version in the browser:

Chapter 6 A Large-Scale Image Database of Malicious Software.

Scott Freitas, Rahul Duggal, Duen Horng Chau. [Submitting to] *Knowledge Discovery and Data Mining, 2022*. <https://arxiv.org/abs/2102.01072>

CHAPTER 5

A LARGE-SCALE DATABASE FOR GRAPH REPRESENTATION LEARNING

With the rapid emergence of graph representation learning, the construction of new large-scale datasets is necessary to distinguish model capabilities and accurately assess the strengths and weaknesses of each technique. By carefully analyzing existing graph databases, we identify 3 critical components important for advancing the field of graph representation learning: (1) large graphs, (2) many graphs, and (3) class diversity. To date, no single graph database offers all these desired properties. We introduce MALNET-GRAPH, the largest public graph database ever constructed, representing a large-scale ontology of malicious software function call graphs. MALNET-GRAPH contains over 1.2 million graphs, averaging over 15k nodes and 35k edges per graph, across a hierarchy of 47 types and 696 families. Compared to the popular REDDIT-12K database, MALNET-GRAPH offers **105× more graphs**, **39× larger graphs** on average, and **63× more classes**. We provide a detailed analysis of MALNET-GRAPH, discussing its properties and provenance, along with the evaluation of state-of-the-art machine learning and graph neural network techniques. The unprecedented scale and diversity of MALNET-GRAPH offers exciting opportunities to advance the frontiers of graph representation learning—enabling new discoveries and research into imbalanced classification, explainability and the impact of class hardness. The database is publicly available at www.mal-net.org.

5.1 Introduction

The emergence of graph data across many scientific fields has led to intense interest in the development of representation learning techniques that encode structured information into low dimensional space for a variety of important downstream tasks (e.g., toxic molecule detection, community clustering, malware detection). However, recent research focusing on developing graph kernels, neural networks and spectral methods to capture graph topology has revealed a number of shortcomings of existing benchmark datasets [179, 180, 181, 182], which often contain graphs that are relatively: (1) limited in number; (2) smaller in scale in terms of nodes and edges; and (3) restricted in class diversity. The state of graph representation benchmarks (e.g., PROTEINS [183], IMDB [184], REDDIT [184]) is analogous to MNIST [185] at its height—a staple of the computer vision community, and often the first dataset researchers would evaluate their methods on. The graph representation community is at a similar inflection point, as it is increasingly difficult for current

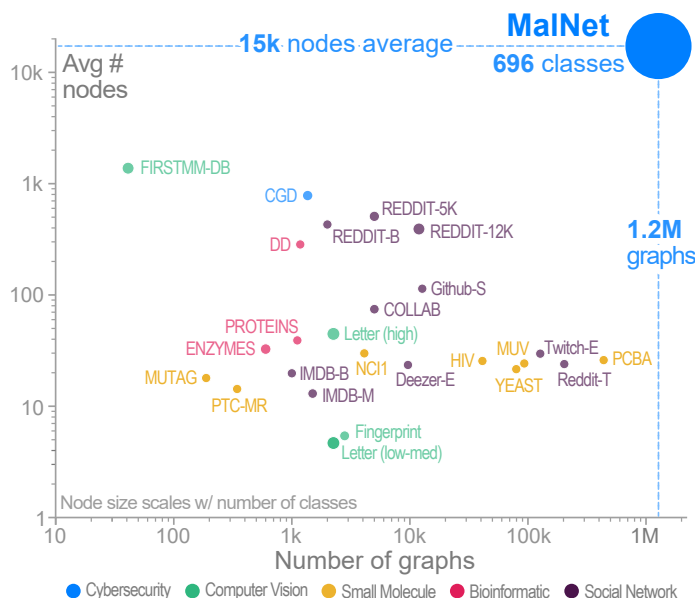


Figure 5.1: MALNET-GRAPH has 1.2M graphs averaging 15k nodes and 35k edges per graph.

databases to characterize and differentiate modern graph representation techniques [179, 180].

To address these issues, we introduce a new graph database called MALNET-GRAPH, a large-scale ontology of malicious software function call graphs (FCGs). Each FCG represents calling relationships between functions in a program, where nodes are functions and edges indicate inter-procedural calls. Through MALNET-GRAPH, we make three major contributions:

- **MALNET-GRAPH: Largest Database for Graph Representation Learning.** MALNET-GRAPH contains 1.2 million function call graphs, averaging over 15k nodes and 35k edges per graph, across a hierarchy of 47 types and 696 families (Figure 5.1). This makes MALNET-GRAPH the largest public graph database constructed to date, offering **105× more graphs**, **39× larger graphs** on average, and **63× more classes** compared to the popular REDDIT-12K database. We release MALNET-GRAPH with a CC-BY license, allowing users to share and adapt the database for any type of use. We also provide code on Github: <https://github.com/safreital/malnet-graph>.
- **Revealing New Discoveries.** The unprecedented scale of MALNET-GRAPH enables new and important discoveries that were previously not possible. Leveraging the function call graphs in MALNET-GRAPH, we study popular graph representation learning techniques in depth, and reveal: (1) the significant challenges they face in terms of scalability and their ability to handle large class imbalance and (2) that simple baselines can be surprisingly effective at the scale of MALNET-GRAPH;

Table 5.1: Descriptive statistics for 10 largest graph types. See Table 5.4 for all graph statistics.

Type	# graph	# fams.	Nodes				Edges				Avg. Degree			
			min	mean	max	std	min	mean	max	std	min	mean	max	std
Adware	884K	250	7	14K	211K	16K	4	31K	605K	38K	0.50	2.21	6.24	0.36
Trojan	179K	441	5	15K	228K	18K	4	34K	530K	42K	0.58	2.05	6.74	0.52
Benign	79K	1	5	35K	552K	30K	3	79K	2M	74K	0.58	2.13	5.30	0.31
Riskware	32K	107	5	12K	173K	16K	4	30K	334K	39K	0.58	2.16	5.42	0.56
Addisplay	17K	38	37	13K	98K	15K	37	28K	246K	34K	0.92	1.97	4.38	0.37
Spr	14K	46	12	28K	169K	21K	7	67K	369K	52K	0.58	2.27	4.70	0.44
Spyware	7K	19	12	5K	55K	6K	7	11K	121K	14K	0.58	1.95	4.27	0.46
Exploit	6K	13	19	24K	102K	14K	14	45K	250K	30K	0.74	1.88	3.34	0.33
Downloader	5K	7	37	20K	107K	28K	37	46K	321K	63K	0.96	1.68	3.53	0.66
Smssend++Trojz	4K	25	16	34K	147K	19K	13	82K	387K	48K	0.81	2.39	3.78	0.23

- **Enabling New Research Directions.** MALNET-GRAPH offers unique opportunities to advance the frontiers of graph representation learning by enabling research into *imbalanced classification*, *explainability* and the impact of *class hardness*. We believe the diversity, scale and natural imbalance of MALNET-GRAPH will enable it to become a benchmark dataset to meet the future research needs of the graph representation community. By open-sourcing MALNET-GRAPH, we hope to inspire and invite more researchers to contribute to this exciting new resource.

5.2 Properties of MalNet

We begin by analyzing 5 key properties of the MALNET-GRAPH database—(1) *scale* (number of graphs, average graph size), (2) class *hierarchy* (3) class *diversity*, (4) *class imbalance* and (5) *cybersecurity applications*. In Section 5.2.1 we compare MALNET-GRAPH against common graph classification datasets, summarizing the differences in Table 5.2.

Scale. MALNET-GRAPH contains 1,262,024 function call graphs across 47 types and 696 families of malware. When stored on disk, MALNET-GRAPH takes over 443 GB of space in edge list format, with each graph containing 15,378 nodes and 35,167 edges, on average. This makes MALNET-GRAPH the **largest public graph dataset constructed** to date in terms of *number of graphs*, *average graph size* and *number of classes*. In Table 5.1, we provide descriptive statistics on the number of nodes, edges, and average degree of ten of the largest graph types (see Table 5.4 for a full comparison). We believe that this scale of data is crucial to the future development of graph representation techniques as current databases are too small to effectively differentiate and benchmark techniques on non-attributed graphs [179, 180, 181, 182].

Hierarchy. Function call graphs are assigned a general *type* (e.g., Worm) and specialized

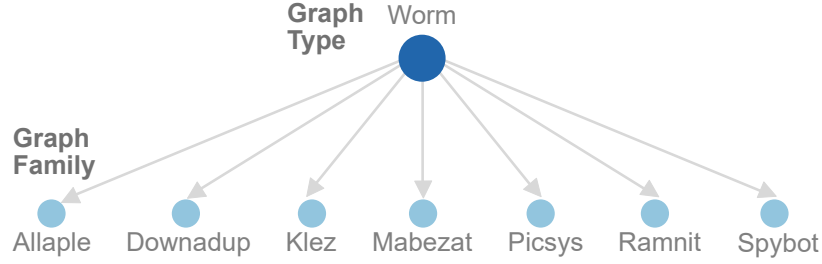


Figure 5.2: Example of the graph type “worm” and its 7 families.

family label (e.g., Spybot) using the Euphony [186] classification structure (see Figure 5.2). To generate these labels, Euphony takes a VirusTotal [187] report containing up to 70 labels across a variety of antivirus vendors and unifies the labeling process by learning the patterns, structure and lexicon of vendors over time. While Euphony provides state-of-the-art performance, this task is considered an open-challenge due to both naming disagreements [188, 189] and a lack of adopted naming standards [186] across vendors. To help address this issue, we collect and release the raw VirusTotal reports containing up to 70 antivirus labels for each graph.

Diversity & Imbalance. MALNET-GRAPH offers 47 types and 696 families of function call graphs following a long tailed distribution with imbalance ratios of $7,827\times$ and $16,901\times$, respectively. To put this in perspective, MALNET-GRAPH’s smallest class contains only 113 samples of the *Click* graph, while 884,455 of the *Adware* type. Models learning from long-tailed distributions tend to favor the majority class, leading to poor generalization performance on rare classes. While class imbalance is traditionally solved by resampling the data (undersampling, oversampling) [190, 191], reshaping the loss function (loss reweighting, regularization) [192, 193] or accounting for input-hardness [194], it is largely unexplored in the graph domain. We hope that MALNET-GRAPH can serve as a source of data to spark novel research in this critical area.

Cybersecurity Applications. A majority of malware samples are *polymorphic* in nature, meaning that subtle source code changes in the original malware variant can result in significantly different compiled code (e.g., instruction reordering, branch inversion, register allocation) [8, 9]. Cybercriminals frequently take advantage of this to evade signature based detection, a predominant form of malware detection [10]. Fortunately, these subtle source code changes have minimal effect on the control flow of the executable, which can be represented with a *function call graph* (see Figure 5.3). Research has demonstrated that function call graphs (FCGs) can effectively defeat the polymorphic nature of malware through techniques like graph matching [11, 12, 13, 14, 15] and representation learning [16, 17]. Unfortunately, prior to the release of MALNET-GRAPH, no large-scale FCG datasets

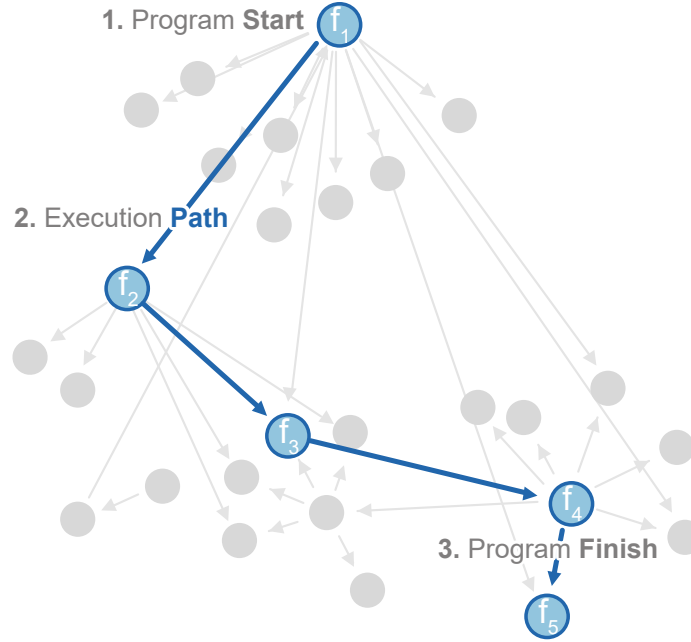


Figure 5.3: FCG from the *Banker++Trojan* type, and *Acecard* family. Nodes represent functions and edges indicate inter-procedural calls. Highlighted in blue is *one* potential execution path.

have been made publicly available largely due to the proprietary nature of the data. We note that while open research can significantly advance the frontiers of cybersecurity, it can be used by malicious actors to conduct research on detection avoidance.

5.2.1 Graph Representation Learning Databases: Advancing the State-of-the-Art

A number of well labeled small datasets have served as training and evaluation benchmarks for most of today’s graph representation learning techniques. As the field advances, larger and more challenging datasets are needed for the next generation of algorithms. MALNET-GRAPH offers **105× more graphs**, **39× larger graphs** on average, and **63× the classes**, compared to the popular REDDIT-12K database. We compare MALNET-GRAPH with other graph representation learning datasets and summarize the differences in Table 5.2, highlighting how MALNET-GRAPH advances the field of graph representation learning by providing large and diverse data.

Cybersecurity datasets. Aside from MALNET-GRAPH, CGD [18] is the only publicly available cybersecurity dataset we could identify for the task of graph classification. In surveying the extensive FCG malware detection literature [11, 16, 12, 17, 13, 14, 15] we observed that almost all data is closed-source; likely due to a combination of security concerns and issues regarding private company data.

Small molecule datasets. There are numerous small molecule datasets, including: HIV [198],

Table 5.2: Comparison of MALNET-GRAPH properties with common graph classification datasets. MALNET-GRAPH offers over 1.2 million graphs averaging 15k nodes and 35k edges with a hierarchical class structure containing 47 types and 696 families. This makes MALNET-GRAPH the largest public graph database constructed to date, offering 105× more graphs, 39× larger graphs on average, and 63× more classes compared to the popular REDDIT-12K database. CC is the clustering coefficient.

Application	Dataset	Graphs	Classes	Ratio	Avg. Node	Avg. Edge	Avg. Degree	Avg. CC	Hierarchy
Cyber	MALNET-GRAPH	1,262,024	696	16,901	15,378	35,167	4.34	.029	✓
	CGD [18]	1,361	2	1.49	782	1852	4.33	.095	-
Small molecule	PCBA [195]	437,929	2	-	26	56	2.16	.002	-
	MUV [196]	93,087	2	-	24	53	2.16	.001	-
	YEAST [197]	79,601	2	1.26	22	23	2.09	.002	-
	HIV [198]	41,127	2	-	26	55	2.15	.002	-
	NCI1 [199]	4,110	2	1	30	32	2.16	.003	-
	PTC-MR [200]	344	2	1.26	14	15	1.98	.009	-
	MUTAG [201]	188	2	1.98	18	20	2.19	.000	-
	Fingerprint [202]	2,800	4	276.5	5	4	1.14	.001	-
Computer vision	Letter-low [202]	2,250	15	1	5	3	1.32	.000	-
	Letter-med [202]	2,250	15	1	5	5	1.35	.014	-
	Letter-high [202]	2,250	15	1	45	5	1.89	.298	-
	FIRSTMM-DB [203]	41	11	3	1377	3074	4.50	.263	-
	DD [204]	1,178	2	1.42	284	716	4.98	.479	-
Bioinfo.	PROTEINS [183]	1,113	2	1.47	39	73	3.73	.514	-
	ENZYMES [183]	600	6	1	33	62	3.86	.453	-
	Reddit-T [205]	203,088	2	15	24	12	2.01	.047	-
Social network	Twitch-E [205]	127,094	2	1.16	30	72	5.39	.549	-
	Github-S [205]	12,725	2	1.15	114	117	3.19	.191	-
	REDDIT-12K [184]	11,929	11	5.05	391	457	2.28	.033	-
	Deezer-E [205]	9,629	2	1.32	23	33	4.29	.510	-
	COLLAB [4]	5,000	3	3.35	74	2458	37.37	.891	-
	REDDIT-5K [184]	4,999	5	1	509	595	2.25	.027	-
	REDDIT-B [184]	2,000	2	1	430	498	2.34	.048	-
	IMDB-M [184]	1,500	3	1	13	66	8.10	.969	-
	IMDB-B [184]	1,000	2	1	20	97	8.89	.947	-

MUTAG [201], MUV [196], PCBA [195], NCI1 [199], PTC-MR [200], and YEAST [197]. The HIV dataset, introduced by the Drug Therapeutics Program AIDS Antiviral Screen [206], tests the ability of chemical compounds to inhibit HIV replication into one of three classes. MUTAG contains chemical compound graphs divided into two classes according to their mutagenic effect on bacterium. MUV and PCBA are constructed from PubChem BioAssay [207], and contain numerous compounds across 17 tasks and 128 tasks respectively, where each task is a binary classification problem. NCI1 contains chemical compounds, screened for their ability to inhibit the growth of a panel of human tumor cell lines. PTC-MR contains graphs across 2 classes, reporting the effects of chemical compound carcinogenicity on rats. YEAST contains molecule graphs screened for anti-cancer tests, with the

binary classification of active or inactive.

Bioinformatic datasets. Three popular bioinformatic datasets are: DD [204], ENZYMES [183] and PROTEINS [183]. DD is a data set containing protein structures grouped into 2 categories (enzyme and non-enzyme). ENZYMES contains graphs of protein tertiary enzyme structures with the task of assigning each enzyme to one of 6 levels. Similarly, *PROTEINS* contains protein graphs classified into either enzyme or non-enzyme.

Computer vision datasets. Three common computer vision datasets are: Fingerprint [202], FIRSTMM-DB [203] and Letter (low, med, high) [202]. Fingerprint contains fingerprint graphs across four classes: arch, left, right, and whorl. FIRSTMM-DB contains object point clouds belonging to an object ontology of 11 categories. Letter contains 3 datasets and 15 character classes with varying levels of distortion (low, med, high) added to letter graphs.

Social network datasets. Common social network datasets include: COLLAB [4], Deezer Ego-Nets [205], Github Stargazers [205], IMDB (BINARY, MULTI) [208], REDDIT (BINARY, 5K, 12K, Threads) [208, 205] and Twitch Ego-Nets [205]. COLLAB is a collaboration dataset of ego-networks across 3 domains of physics. Deezer Ego-Nets contains user ego-nets across 2 genders from the Deezer music service. Github Stargazers contains graphs of developers who starred either machine learning or web development repositories. IMDB BINARY contains ego-network graphs representing actors and their collaborations across 2 movie genres. IMDB MULTI extends IMDB BINARY with more graphs and 3 movie genres. REDDIT-BINARY contains thread graphs across two content classes (discussion and QA based). REDDIT MULTI-5K contains thread graphs across 5 Reddit thread types. REDDIT MULTI-12K extends REDDIT-5K, containing online discussion thread graphs across 11 classes. REDDIT Threads contains thread graphs across 2 graph classes (discussion, non-discussion). Twitch Ego-Nets contains ego graphs across 2 classes of Twitch users.

5.3 Constructing MalNet

5.3.1 Collecting Candidate Graphs

The first step in constructing MALNET-GRAPH was to identify a source of graph containing the desired properties outlined in Section 5.2. We determined that the natural abundance, large graph size, and class diversity provided by function call graphs (FCGs) make them an ideal source of graphs. While FCGs, which represent the control flow of programs (see Figure 5.3), can be statically extracted from many types of software (e.g., EXE, PE, APK),

we use the Android ecosystem due to its large market share [209], easy accessibility [210] and diversity of malicious software [211]. With the generous permission of the AndroZoo repository [212, 210], we collected 1,262,024 Android APK files, specifically selecting APKs containing both a *family* and *type* label obtained from the Euphony classification structure [186]. This process took about a week to download and 10TB in storage space when using the maximum allowed 40 concurrent downloads. In addition, we spent about 1 month collecting raw VirusTotal (VT) reports to release with MALNET-GRAPH, through VT’s academic access, which allows 20k queries per day. Each VT report contains up to 70 antivirus labels per graph.

5.3.2 Processing the Graphs

Once the APK files and labels were collected, we extract the function call graphs by running the files through Androguard [213], which statically analyzes the APK’s DEX file. Distributed across Google Clouds General-purpose (N2) machine with 16 cores running 24 hours a day, the process took about 1 week to extract the graphs. We leave each graph in its original state—retaining its edge directionality, disconnected components and node isolates (i.e., single nodes with no incident edges). On average, each graph has 15,378 nodes and 35,167 edges; and typically contains a single giant connected component, many small disconnected components, and numerous node isolates. Table 5.1 describes the 10 graph types (out of 47) that have the highest number of graphs. Table 5.4 provides a full analysis on all graph type. Each graph is stored in a standard edge list format for its wide support, readability, and ease of use. In total, the graphs’ edge list files consume over 443 GB of hard disk space. Since we are dealing with highly malicious software, our goal is to mitigate the risk of releasing information that could potentially be used to reverse engineer malware. Thus, we numerically relabel the nodes of each graph, removing any associated attribute information, which makes reverse engineering highly unlikely. However, malicious actors could develop new variants of detection-resistant malware that looks structurally similar to benign function call graphs, by gleaning graph structure knowledge from MALNET-GRAPH in the absence of node and edge labels.

5.3.3 MalNet-Tiny

We construct MALNET-GRAPH-TINY, containing 5,000 graphs across balanced 5 *types*. In addition, we limit each graph to contain at most 5k nodes so that the dataset is truly “tiny”. The goal of MALNET-GRAPH-TINY is to enable users to rapidly prototype new ideas, since it requires only a fraction of the time needed to train a new model. MALNET-

GRAPH-TINY is released alongside the full dataset at <https://mal-net.org>.

5.3.4 Online Exploration of the Data

To assist researchers and practitioners in exploring MALNET-GRAPH, we have designed and developed MALNET-GRAPH EXPLORER, an interactive graph exploration and visualization tool. It runs on most modern web browsers (Chrome, Firefox, Safari, and Edge), platforms (Windows, Mac OS, Linux), and devices (Android and iOS). Our goal is to enable users to easily explore the data before downloading. MALNET-GRAPH EXPLORER’s user interface uses a *responsive* design that automatically adjusts its component layout, based on the users’ device types and screen resolutions. MALNET-GRAPH EXPLORER is available online at: <https://mal-net.org>.

5.4 MalNet for New Research & Discoveries

MALNET-GRAPH is substantially larger than existing graph databases used for graph representation learning research, with many more graphs, much larger graphs, and many more classes of graphs. Such unprecedented advancements provides exciting opportunities to make new discoveries and explore new research directions previously not possible. In this section, we present our findings to demonstrate such possibilities. We discuss the experimental setup below, followed by an overview of the graph representation techniques in Section 5.4.1. Section 5.4.2 discusses the new discoveries we found by studying MALNET-GRAPH; and Section 5.4.3 highlights new research directions enabled by MALNET-GRAPH.

Experimental Setup. We divide MALNET-GRAPH into three stratified sets of data: training, validation and test, with a split of 70/10/20, respectively; repeated for graph *type*, *family* and MALNET-TINY labels. Each model is evaluated on its macro-F1 score, however, we report three performance metrics—macro-F1, precision and recall, as is typical for highly imbalanced datasets [194, 214]. We perform our experiments in Python3 using a DGX A-100 containing 128 CPU cores and 8 A-100 GPUs.

5.4.1 Graph Representation Techniques

We present results for 7 strong, recent, scalable, and readily available graph representation techniques [215, 205]. Specifically, we evaluate 2 graph neural network (GNN) models [216, 217] and 5 data mining techniques [179, 181, 218, 219]. We leave the graph in its natural state for each GNN i.e., directed graph with isolates; and follow recommended

preprocessing steps from the paper of each data mining technique. In addition, each data mining embedding techniques uses a random forest model for the task of graph classification, where we run a grid search across the validation set to identify the number of estimators $n_e \in [1, 5, 10, 25, 50]$ and tree depth $t_d \in [1, 5, 10, 20]$. All hyperparameters are individually tuned for *type*, *family* and *tiny* classification levels. We briefly summarize each method and its configuration below:

1. **GCN** [216] is a graph neural network which learns network embeddings by aggregating node features over neighborhoods. Following [217], we use 5 GNN layers and an Adam optimizer [220]. We set node features using LDP [179], and add self loops which has been shown to improve performance [221]. We tune hyperparameters for (1) the number of hidden units $\in \{32, 64\}$ and (2) the learning rate $\in \{0.001, 0.0001\}$, repeated for both *type* and *family* classification levels. We find that 64 units with a learning rate of 0.0001 performs best. Running this search took over **26 days** using the Nvidia DGX A100, their most powerful commercial GPU server.
2. **GIN** [217] is a state-of-the-art GNN with strong theoretical backing. Following [217], we set $\epsilon = 0$, use 5 GNN layers, and an Adam optimizer [220]. We set node features using LDP [179], and add self loops which has been shown to improve performance [221]. We tune hyperparameters for (1) the number of hidden units $\in \{32, 64\}$ and (2) the learning rate $\in \{0.001, 0.0001\}$. We find that 64 units with a learning rate of 0.0001 performs best. Running this search took over **23 days** using the Nvidia DGX A100.
3. **LDP** [179] is a simple representation scheme that summarizes each node and its 1-hop neighborhood using using 5 degree statistics. These node features are then aggregated into a histogram where they are concatenated into feature vectors. We use the parameters suggested in [179]. Running this method took 4 hours parallelized across all 128 CPU cores of the Nvidia DGX A100.
4. **NoG** [181] ignores the topological graph structure, viewing the graph as a two-dimensional feature vector of the node and edge count. Running this method took approximately 1 hour parallelized across all 128 CPU cores of the Nvidia DGX A100.
5. **Feather** [218] is a more complex representation scheme that uses characteristic functions of node features with random walk weights to describe node neighborhoods. We perform a search over the key *order* $\in \{4, 5, 6\}$ parameter, which controls how much information is seen from higher order neighborhoods. We find that an order of 5 performs best. For the remaining parameters, we use the values suggested in [218]. Running this search took over 19 hours parallelized across all 128 CPU cores of the Nvidia DGX A100.

6. **Slaq-VNGE** [219] approximates the spectral distances between graphs based on the Von Neumann Graph Entropy (VNGE), which measures information divergence and distance between graphs [222]. We perform a search over 2 key parameters: number of random vectors $n_v \in \{10, 15, 20\}$ and the number of Lanczos steps $s \in \{10, 15, 20\}$. We find that $n_v = 15$ and $s = 15$ performs best. For the remaining parameters, we use the values suggested in [219]. Running this search took 8 hours parallelized across all 128 CPU cores of the Nvidia DGX A100.
7. **Slaq-LSD** [219] approximates NetLSD, which measures the spectral distance between graphs based on the heat kernel [223]. We perform a search over 2 key parameters: number of random vectors $n_v \in \{10, 15, 20\}$ and number of Lanczos steps $s \in \{10, 15, 20\}$. We find that $n_v = 20$ and $s = 20$ performs best. For the remaining parameters, we use the values suggested in [219]. Running this search took 8 hours parallelized across all 128 CPU cores of the Nvidia DGX A100.

Limitations. We tested a number of alternative graph representation techniques and decided to exclude them—methods based on kernel [224, 225, 226, 226], spectral [227, 223, 228, 229, 230] and document embedding [231, 232]—as they were computationally prohibitive for the scale of MALNET-GRAPH, making it infeasible to run the techniques over the full dataset or perform parameter selection. We also note that methods that work well on other datasets may not work well on MALNET-GRAPH due to its larger scale and different structural properties (see Table 6.1); vice-versa, methods that work on MALNET-GRAPH may not transfer well to other datasets. We hope MALNET-GRAPH will inspire the release of additional large-scale datasets in the call graph domain and other novel application areas, which will help enable researchers to develop and evaluate methods that generalize across domains.

5.4.2 Enabling New Discoveries

Current graph representation research uses datasets that are significantly smaller in scale, and much less diverse compared to MALNET-GRAPH. In light of this, we want to study what new discoveries can be made, that were previously not possible due to dataset limitations. For example, what is the impact of class imbalance and diversity in the classification process? We synthesized our findings into the following 2 major discoveries (D1-D2).

D1. Less Diversity, Better Performance. Comparing methods in Table 5.3 across malware *type* (low diversity, with 47 classes) and *family* (high diversity, with 696 classes), the classification task becomes increasingly difficult as diversity and data imbalance increase. This trend is visible across all 7 graph representation methods. For the best

Table 5.3: Comparison of macro-F1, precision and recall scores achieved by 7 methods at the *type* (low diversity, with 47 classes) and *family* (high diversity, with 696 classes) and *tiny* (5k graphs across 5 balanced classes) classification levels. Comparing methods across *type* and *family*, the classification task becomes increasingly difficult as diversity and data imbalance increase.

Method	Type			Family			TINY
	Macro-F1	Precision	Recall	Macro-F1	Precision	Recall	Accuracy
Feather [218]	.41	.71	.35	.34	.56	.29	.86
LDP [179]	.38	.69	.31	.34	.55	.28	.86
GIN [217]	.39	.57	.36	.28	.32	.28	.90
GCN [216]	.38	.51	.35	.21	.24	.21	.81
Slaq-LSD [219]	.33	.62	.26	.24	.42	.19	.76
NoG [181]	.30	.62	.25	.25	.42	.21	.77
Slaq-VNGE [219]	.04	.07	.04	.01	.01	.01	.53

performing method, Feather, the macro-F1 score drops from 0.41 (*type*) to 0.34 (*family*). This matches our intuition from the “tiny” experiments in Table 5.3, which shows strong method performance when evaluating on a small subset of MALNET-GRAPH, containing 5 well-balanced types.

D2. Simple Baselines Surprisingly Effective. Both NoG and LDP use basic graph statistics. Given the simplicity of these methods, they perform remarkably well, often outperforming or matching the performance of more complex methods. For example, in Table 5.3 we can see that LDP ties for the best performing *family* classification method, achieving a macro-F1 score of 0.34, while beating significantly more complex methods e.g., GIN, GCN, Slaq-LSD. A similar trend is found in *type* level classification results where LDP outperforms SLAQ-LSD and performs on par with GIN and GCN, despite being simpler and significantly faster than all 3 methods. Using small graph databases, earlier work [179] suggested the potential merits of considering simpler approaches. For the first time, using the largest graph database to date, our result confirms that many current techniques in the literature do not well capture non-attributed graph topology.

5.4.3 Enabling New Research Directions

The unprecedented scale and diversity of MALNET-GRAPH opens up new exciting research opportunities for the graph representation community. Below, we present four promising directions (R1-R4).

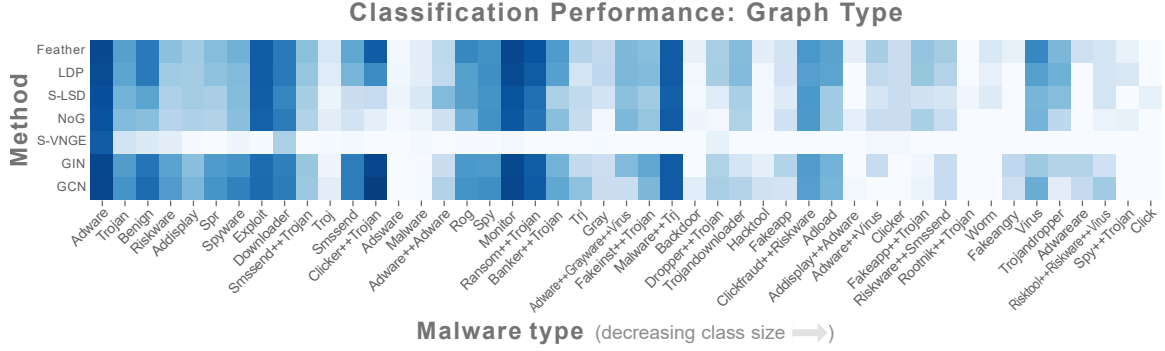


Figure 5.4: Class-wise comparison of model predictions where a darker cell represents a higher F1 score. We observe that certain classes are more challenging to classify than others.

R1. Class Hardness Exploration. Because of MALNET-GRAPH’s large diversity, it is now possible for researchers to explore why certain classes are more challenging to classify than others. For example, Figure 5.4 shows *Malware++Trj* significantly outperforming both *Troj* and *Adware*, which contain many more examples. This result is surprising, and provides strong impetus for additional research into class hardness, such as: (a) investigating whether existing methods are flexible enough to represent the diverse graph structures; and (b) inviting researchers to study the similarities across class types (e.g. merge *Spr* and *Spyware*). To support further development in this challenging area, we release the raw VirusTotal reports containing up to 70 labels per graph.

R2. Imbalanced Classification Research. The natural world often follows a long-tailed data distribution where only a few classes account for most of the examples [194]. As evidenced in discovery **D1**, the long-tail often causes classifiers to perform well on the majority class, but poorly on rare ones. Unfortunately, imbalanced classification research in the graph domain has yet to receive much attention, largely because no datasets existed to support the research. By releasing MALNET-GRAPH, the largest naturally imbalanced database to date, we hope foster new interest in this important area.

R3. Reconsidering Merits of Simpler Graph Classification Approaches. Our discovery in **D2** indicates that simpler methods can match or outperform more recent and sophisticated techniques, suggesting that current techniques aiming to capture graph topology are not yet well-reflected for non-attributed graphs, echoing results from [179]. More broadly, our discovery demonstrates—for the first time—such phenomenon at the unprecedented scale and diversity offered by MALNET-GRAPH. We believe our results will inspire researchers to reconsider the merits of simpler approaches and

classic techniques, and to build on them to reap their benefits.

R4. Enabling Explainable Research. In Figure 5.4, we observe that certain representation techniques better capture particular graph types. For example, Feather, GIN and GCN significantly outperforms other methods on *Clicker++Trojan*. This is an interesting result, as it could provide insight into when one technique is preferred over another (e.g., local neighborhood structure, global graph structure, graph motifs). We believe that the wide range of graph topology and substructures contained in MALNET-GRAPH’s nearly 700 classes will enable new explainability research.

5.5 Conclusion

The study of graph representation learning is a critical tool in the characterization and understanding of complex interconnected systems. Currently, no large-scale database exists to accurately assess the strengths and weaknesses of these techniques. To address this, we contribute a new large-scale database—MALNET-GRAPH—containing 1,262,024 graphs, averaging over 15k nodes and 35k edges per graph, across a hierarchy of 47 types and 696 families. We hope MALNET-GRAPH will become a central resource for a broad range of graph research. The database is available at www.mal-net.org.

Type	# graphs	# fams.	Nodes				Edges				Avg. Degree			
			min	mean	max	std	min	mean	max	std	min	mean	max	std
Adware	884K	250	7	14K	211K	16K	4	31K	605K	38K	0.50	2.21	6.24	0.36
Trojan	179K	441	5	15K	228K	18K	4	34K	530K	42K	0.58	2.05	6.74	0.52
Benign	79K	1	5	35K	552K	30K	3	79K	2M	74K	0.58	2.13	5.30	0.31
Riskware	32K	107	5	12K	173K	16K	4	30K	334K	39K	0.58	2.16	5.42	0.56
Addisplay	17K	38	37	13K	98K	15K	37	28K	246K	34K	0.92	1.97	4.38	0.37
Spr	14K	46	12	28K	169K	21K	7	67K	369K	52K	0.58	2.27	4.70	0.44
Spyware	7K	19	12	5K	55K	6K	7	11K	121K	14K	0.58	1.95	4.27	0.46
Exploit	6K	13	19	24K	102K	14K	14	45K	250K	30K	0.74	1.88	3.34	0.33
Downloader	5K	7	37	20K	107K	28K	37	46K	321K	63K	0.96	1.68	3.53	0.66
Smssend++Trojan	4K	25	16	34K	147K	19K	13	82K	387K	48K	0.81	2.39	3.78	0.23
Troj	3K	36	14	6K	64K	8K	11	15K	115K	18K	0.79	1.98	5.60	0.52
Smssend	3K	12	15	20K	111K	14K	12	49K	337K	38K	0.80	2.34	4.61	0.47
Clicker++Trojan	3K	3	220	6K	29K	3K	471	14K	72K	7K	1.52	2.33	2.92	0.18
Adware	3K	16	368	11K	53K	13K	564	26K	143K	28K	1.02	2.19	4.27	0.26
Malware	3K	19	6	8K	119K	13K	5	16K	286K	29K	0.83	1.90	3.97	0.67
Adware++Adware	3K	2	192	9K	55K	6K	289	20K	138K	16K	1.49	2.16	3.17	0.27
Rog	2K	22	26	15K	102K	19K	31	35K	232K	46K	0.91	2.05	4.79	0.49
Spy	2K	7	48	22K	107K	15K	44	49K	271K	40K	0.92	2.17	3.07	0.25
Monitor	1K	5	329	4K	41K	5K	580	7K	102K	12K	1.53	1.83	3.09	0.21
Ransom++Trojan	1K	7	556	51K	139K	22K	965	115K	319K	48K	1.59	2.26	2.59	0.21
Banker++Trojan	1K	6	29	33K	103K	16K	36	72K	237K	38K	1.22	2.15	2.99	0.24
Trj	940	18	29	13K	171K	16K	36	30K	402K	39K	1.15	2.20	4.44	0.49
Gray	922	10	51	16K	66K	13K	56	39K	153K	31K	0.88	2.09	4.33	0.58
Adware++Grayware++Virus	835	4	22	6K	84K	13K	20	14K	193K	29K	0.86	2.79	3.17	0.34
Fakeinst++Trojan	718	10	51	15K	94K	17K	58	37K	229K	44K	0.99	2.12	2.84	0.48
Malware++Trj	609	1	52K	52K	56K	596	118K	119K	128K	1K	2.28	2.28	2.29	0
Backdoor	602	10	25	13K	146K	22K	21	33K	427K	57K	0.84	2.19	3.55	0.37
Dropper++Trojan	592	8	47	5K	67K	7K	50	11K	175K	18K	1.06	1.98	3.92	0.70
Trojandownloader	568	7	1K	38K	102K	19K	2K	86K	258K	45K	1.34	2.19	2.54	0.21
Hacktool	542	7	668	17K	41K	9K	2K	37K	92K	20K	1.63	2.21	3.64	0.25
Fakeapp	425	5	24	4K	50K	7K	21	8K	107K	16K	0.88	1.67	2.79	0.37
Clickfraud++Riskware	369	5	2K	18K	20K	2K	4K	38K	43K	5K	1.95	2.13	2.25	0.04
Adload	333	4	2K	19K	53K	18K	4K	48K	149K	48K	1.46	2.29	3.13	0.40
Addisplay++Adware	294	1	3K	20K	50K	9K	6K	41K	108K	20K	1.65	2.03	2.45	0.21
Adware++Virus	274	9	38	15K	59K	15K	38	33K	138K	35K	1	2.22	3.17	0.54
Clicker	265	5	47	3K	75K	7K	43	6K	190K	17K	0.91	1.62	3.32	0.51
Fakeapp++Trojan	256	1	44	21K	72K	15K	39	41K	162K	34K	0.88	1.74	2.30	0.27
Riskware++Smssend	247	7	12	2K	60K	6K	7	5K	154K	14K	0.58	1.68	3	0.45
Rootnik++Trojan	223	5	210	16K	84K	21K	395	39K	197K	50K	1.15	2.59	3.21	0.47
Worm	220	7	64	14K	94K	15K	78	31K	204K	34K	0.99	1.99	3.42	0.40
Fakeangry	211	2	516	6K	98K	11K	946	15K	279K	29K	1.70	2.35	3.29	0.27
Virus	191	3	681	15K	80K	19K	1K	35K	177K	46K	1.32	2.12	3.18	0.33
Trojandropper	178	4	220	20K	78K	18K	236	39K	185K	39K	1.03	1.83	4.36	0.32
Adwareare	152	3	893	26K	57K	14K	2K	60K	144K	32K	1.88	2.25	2.60	0.20
Risktool++Riskware++Virus	152	3	37	16K	65K	16K	37	36K	158K	37K	1	1.92	3.17	0.48
Spy++Trojan	119	5	54	31K	118K	25K	66	75K	293K	61K	1.22	2.31	3.26	0.37
Click	113	1	2K	4K	12K	2K	4K	8K	26K	4K	1.80	2.04	2.74	0.21

Table 5.4: Descriptive statistics for each graph type in MALNET-GRAPH.

CHAPTER 6

A LARGE-SCALE IMAGE DATABASE OF MALICIOUS SOFTWARE

Computer vision is playing an increasingly important role in automated malware detection with the rise of the image-based binary representation. These binary images are fast to generate, require no feature engineering, and are resilient to popular obfuscation methods. Significant research has been conducted in this area, however, it has been restricted to small-scale or private datasets that only a few industry labs and research teams have access to. This lack of availability hinders examination of existing work, development of new research, and dissemination of ideas. We release MALNET-IMAGE, the **largest public cybersecurity image database**, offering **24× more images** and **70× more classes** than existing databases (available at <https://mal-net.org>). MALNET-IMAGE contains over 1.2 million malware images—across 47 types and 696 families—democratizing image-based malware capabilities by enabling researchers and practitioners to evaluate techniques that were previously reported in propriety settings. We report the first million-scale malware detection results on binary images. MALNET-IMAGE unlocks new and unique opportunities to advance the frontiers of machine learning, enabling new research directions into vision-based cyber defenses, multi-class imbalanced classification, and interpretable security.

6.1 Introduction

Attack campaigns from criminal organizations and nation state actors are one of the most powerful forms of disruption, costing the U.S. economy as much as \$109 billion a year [242]. These cyber attacks are highly sophisticated, targeting governments and large-scale enterprises to interrupt critical services and steal intellectual property [59]. Defending against these attacks requires the development of strong antivirus tools to identify new variants of malicious software before they can infect a network. Unfortunately, as a majority of newly identified malware is *polymorphic* in nature, where a few subtle source code changes result in significantly different compiled code (e.g., instruction reordering, branch inversion, register allocation) [9, 8], the predominant signature-based form of malware detection is rendered inert [10].

To combat these issues, the cybersecurity industry [235] has turned to image-based malware representations as they are quick to generate, require no feature engineering, and are resilient to common obfuscation techniques (e.g., section encryption [243]). For all

Table 6.1: MALNET-GRAPH has 1.2M images across a hierarchy of 47 types and 696 families.

	Dataset	Images	Classes
Public	MALNET-GRAPH	1,262,024	696
	Virus-MNIST [233]	51,880	10
	Malimg [234]	9,458	25
Private	Stamina [235]	782,224	2
	McAfee [236]	367,183	2
	Kancherla [237]	27,000	2
	Choi [238]	12,000	2
	Fu [239]	7,087	15
	Han [240]	1,000	50
	IoT DDoS [241]	365	3

of these reasons, image-based malware detection and classification research has surged in popularity. Unfortunately, a majority of this research uses small-scale or private data repositories, making it increasingly difficult to characterize and differentiate existing work, develop new research methodologies, and disseminate new ideas [235, 244, 245, 239, 236, 240, 246, 247, 243, 234, 248]. To address these issues, we constructed MALNET-IMAGE, the first large-scale ontology of malicious software images. Through MALNET-IMAGE, we make three major contributions:

- **MALNET-IMAGE: Largest Cybersecurity Image Database.** MALNET-IMAGE is the largest public cybersecurity image database, containing over 1.2 million software images across a hierarchy of 47 types and 696 families. Compared to the next large public database [233], MALNET-IMAGE offers $24\times$ more images and nearly $70\times$ more classes (see Table 6.1). By releasing the first database of its kind, MALNET-IMAGE enables new and important discoveries in malware detection and classification research that was previously restricted to a few industry labs and research teams. We release MALNET-IMAGE with a CC-BY license, enabling researchers and practitioners to share and adapt the database according to their needs. We release the code containing experiments and dataset creation on Github: <https://github.com/safreital/malnet-image>.
- **Democratizing Image-Based Malware Capabilities.** Researchers and practitioners can now conduct experiments on an industry scale dataset, evaluating techniques that were previously reported in propriety settings. We report the first public large-scale malware

detection results on binary images, where we are able to identify malicious files with an AUC of 0.94. MALNET-IMAGE also enables new research into multi-class malware classification using binary-images (e.g., is this Ransomware or Spyware?), a critical tool in formulating a defense response. Our first of their kind results demonstrate that a ResNet18 model can classify 47 *types* and 696 *families* of malware with a macro-F1 score of 0.49 and 0.45, respectively.

- **Enabling New Research Directions.** MALNET-IMAGE offers new and unique opportunities to advance the frontiers of cybersecurity research. In particular, MALNET-IMAGE offers researchers a chance to study imbalanced classification on a large-scale cybersecurity database with a natural imbalance ratio of $16,901\times$; and explore explainability research in a high impact domain, where it is critical that security analysts can interpret and trust the model.

6.2 Properties of MalNet-Image

We begin by analyzing 5 key properties of the MALNET-IMAGE database—(1) size, (2) labeling (3) data diversity, (4) class imbalance and (5) cybersecurity applications.

Size. MALNET-IMAGE is the largest cybersecurity image dataset ever released, containing 1,262,024 binary images across 47 types and 696 families of malware. Table 6.2 provides statistics on the number of images and families contained in each type of malware. MALNET-IMAGE offers over $24\times$ more images and nearly $70\times$ more classes than the largest alternative public binary-image database; and 479,800 more images and 694 more classes than Stamina [235], the largest private binary image database. By enabling researchers and practitioners to conduct experiments at an industry scale, MALNET-IMAGE offers exciting new opportunities to develop state-of-the-art malware detection and classification techniques.

Image Labels. Each piece of malware (thus its image representation in MALNET-IMAGE) is assigned both a general malware “type” (e.g., Backdoor), and a specialized malware “family” (e.g., Droidkungfu), using Euphony [186], a state-of-the-art malware labeling system that aggregates and learns from the labelling results of up to 70 antivirus vendors from VirusTotal [187].

Data Diversity & Imbalance. With 47 malware types and 696 malware families, MALNET-IMAGE is one of the most diverse image databases. In Figure 6.1, we can see that the class distribution is highly imbalanced across image *type* and *family*, with imbalance ratios of $7,827\times$ and $16,901\times$, respectively. This long-tailed distribution is a common property of

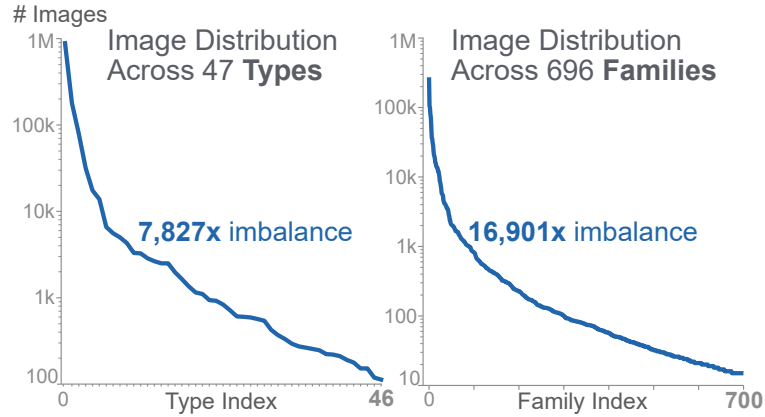


Figure 6.1: *Type* and *family* labels have imbalance ratios of $7,827\times$ and $16,901\times$, respectively.

many real-world datasets, where a few of the classes contain a majority of examples [194]. Table 6.2 provides a breakdown of the number of images and families in each type.

Security Applications. Most newly identified malware samples are packed, meaning that the binary code is obfuscated to evade signature based detection, the predominant form of malware detection [234, 10]. Fortunately, research has shown that image-based binary representations are resilient to common packing techniques since they typically perform a monotonic transformation of the binaries, failing to conceal common byte patterns present in the original binaries [234]. With the release of MALNET-IMAGE, researchers will now have access to a critical resource to develop advanced, image-based malware detection and classification algorithms. Like most open data resources, there is a potential for MALNET-IMAGE to be misused by malicious actors who aim to craft new variants to evade detection. We believe MALNET-IMAGE’s contribution to the research community significantly outweighs such risk.

6.3 MalNet-Image: Advancing the State-of-the-Art

Aside from MALNET-IMAGE, there are only two publicly available binary-image based cybersecurity datasets— Maling [234] and Virus-MNIST [233]—containing 9,458 images across 25 classes, and 51,880 images across 10 classes, respectively. In surveying the malware detection and classification literature [234, 235, 236, 237, 238, 239, 240, 241, 249, 250, 251, 252, 253, 254, 255, 256, 247, 257, 258, 245], we observed that almost all experiments were conducted on small-scale or private data. As the field advances, large-scale public databases are necessary to develop the next generation of algorithms. In Table 6.1, we compare MALNET-IMAGE with other public and private cybersecurity image datasets. We find that that MALNET-IMAGE offers **$24\times$ more images** and

Table 6.2: The number of images and families in each type of malware in MALNET-GRAPH.

Type	# Images	# Families	Type	# Images	# Families
Adware	884K	250	Fakeinst+Trojan	718	10
Trojan	179K	441	Malware+Trj	609	1
Benign	79K	1	Backdoor	602	10
Riskware	32K	107	Dropper+Troja	592	8
Addisplay	17K	38	Trojandownloader	568	7
Spr	14K	46	Hacktool	542	7
Spyware	7K	19	Fakeapp	425	5
Exploit	6K	13	Clickfraud+	369	5
Downloader	5K	7	Riskware		
Smssend+Trc	4K	25	Adload	333	4
Trj	3K	36	Addisplay+Ad	294	1
Smssend	3K	12	Adware+Virus	274	9
Clicker+Trojan	3K	3	Clicker	265	5
Adware	3K	16	Fakeapp+Trojan	256	1
Malware	3K	19	Riskware+Sms	247	7
Adware+Adv	3K	2	Rootnik+Trojan	223	5
Rog	2K	22	Worm	220	7
Spy	2K	7	Fakeangry	211	2
Monitor	1K	5	Virus	191	3
Ransom+Troj	1K	7	Trojandropper	178	4
Banker+Trojan	1K	6	Adwareare	152	3
Trj	940	18	Risktool+Riskware	152	3
Gray	922	10	+Virus		
Adware+	835	4	Spy+Trojan	119	5
Grayware+Vi			Click	113	1

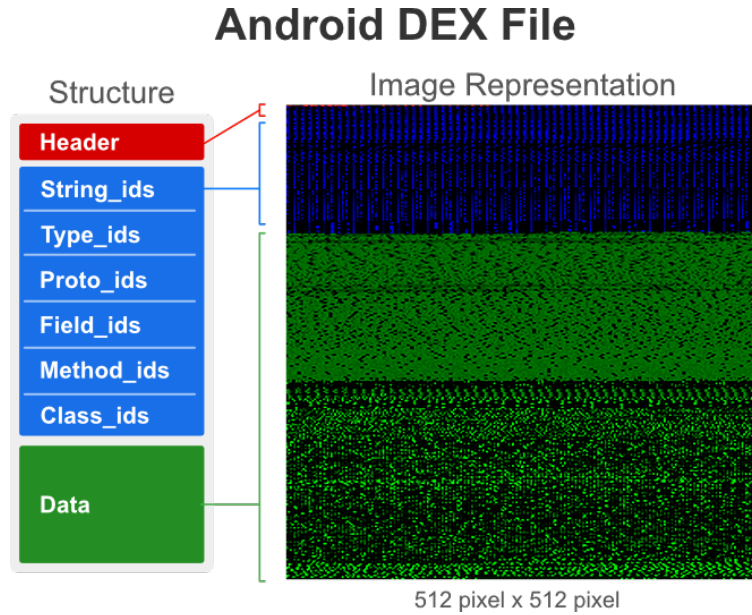


Figure 6.2: **Left:** Android DEX file structure, composed of three major components—(1) header, (2) ids, and (3) data. **Right:** binary image representation of the DEX file.

70× the classes, compared to the largest alternative public binary image database (Virus-MNIST [233]); and 479,800 more images and 694 more classes than the largest private database (Stamina [235]). We do not compare against repositories of malicious binaries such as AndroZoo [210], AMD [208], Microsoft-BIG [259], Malicia [260], VirusShare, and VirusTotal in this discussion, as none of them are readily available to use. To put it in perspective, gathering the labels, downloading and processing the data, and preparing MALNET-IMAGE took months of processing and preparation.

6.3.1 Constructing MalNet-Image

MALNET-IMAGE is an ambitious project to collect and process over 1.2 million binary images, and is a major extension to the graph representation learning database MALNET [261], offering significant new malware detection capabilities. Below, we describe the provenance and construction of MALNET-IMAGE.

Collecting and Processing Candidate Images. We construct MALNET-IMAGE using the Android ecosystem due to its large market share [209], easy accessibility [210] and diversity of malicious software [211]. With the generous permission of the AndroZoo repository [212, 210], we collected 1,262,024 Android APK files, specifically selecting APKs containing both a *family* and *type* label obtained from the Euphony classification structure [186] (APKs available from AndroZoo repository).

Once the APK files and labels were gathered, the first step in constructing the image

Malware Images with Varying "Texture"

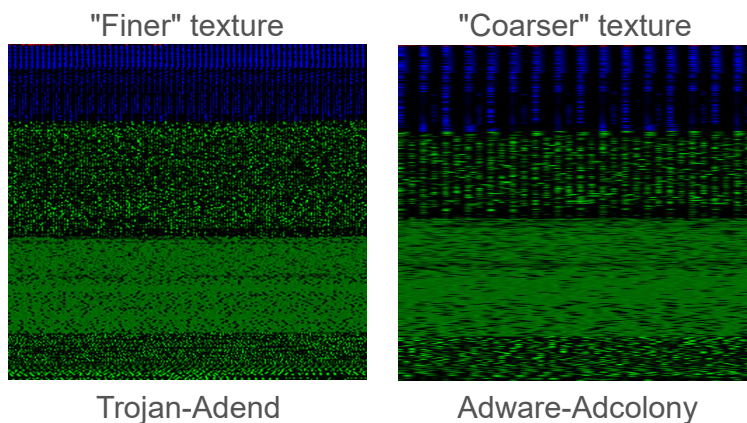


Figure 6.3: Images of two malware types with different “texture”. **Left:** the Trojan image is more “fine-grained”. **Right:** the Adware image is more “coarse”. Malware images belonging to the same type or family often appear visually similar in layout and texture, whereas images across types and families contain noticeable differences in layout and texture.

representation was to extract the DEX file (bytecode) from each Android APK. The extracted DEX file was then converted into a 1D array of 8 bit unsigned integers. Each entry in the array is in the range $[0, 255]$ where 0 corresponds to a black pixel and 255 a white pixel. Once in array form, each binary goes through a 3-stage conversion—(1) converting the 1D array to a 2D image, (2) encoding semantic information into the RGB channels, and (3) scaling the images to a standard size. Distributed across Google Cloud’s General-purpose (N2) machine with 16 cores running 24 hours a day, this process took approximately a week. We release the source code use to process the APKs on Github <https://github.com/safreital/malnet-image>, and describe each step in detail below.

Converting to a 2D representation. We convert the 1D byte array into a 2D array using standard linear plotting where the width of the image is fixed and the height is allowed to vary based on the file size. We use the same image width proposed in the seminal work [243] (and follow up work [262, 263, 264, 265]), and scale each image to 256×256 using a standard Lanczos filter from the Pillow library. In Figure 6.3, we show images of two malware types with different “texture” [243]. On the left, the Trojan image is more “fine-grained”; while on the right, the Adware image is more “coarse”. In addition, each section in the malware image can have a distinctive texture. Looking at the Trojan image (Figure 6.3, left), we can see that the identifier & definitions section (blue) has a unique pattern that repeats vertically, whereas the data section (green) appears more random. Furthermore, the texture within each section can vary, as observed by the 3 distinct subsections within the

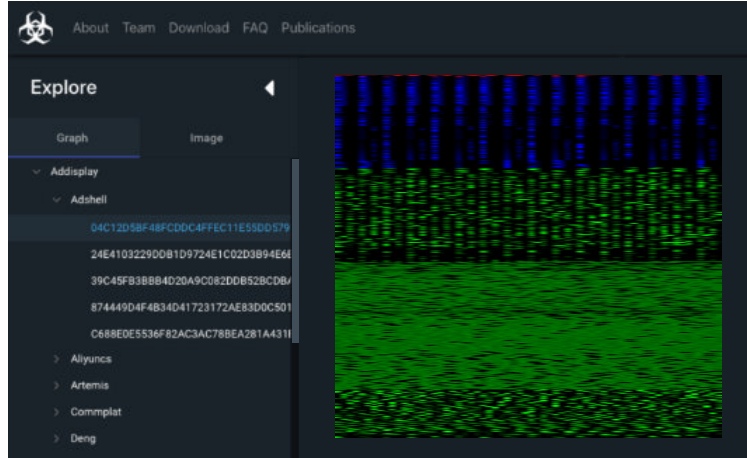


Figure 6.4: MALNET-IMAGE EXPLORER. An exploration panel on the left allows users to select from the available images types and families. Users can then visually the image on the right.

data section of the Trojan and Adware images. We observe that malware images belonging to the same type or family often appear visually similar in layout and texture, whereas images across types and families contain noticeable differences in layout and texture.

Encoding Semantic Information. Semantics can play an important role in analyzing the bytecode of an application. For example, a randomly chosen byte could be an ascii character, opcode or part of a pointer address. By coloring each byte according to its use, the image has an added layer of semantic information on top of the raw bytecode. While a variety of techniques can be used to encode semantic information into the image, there is currently no accepted standard. We follow [236], and encode the semantic information by assigning each byte to a particular RGB color channel depending on its position in the DEX file structure—(i) *header*, (ii) *identifiers* and *class definitions*, and (iii) *data* (see Figure 6.2). To remove this layer of semantic encoding, the images can be converted to grayscale by combining each of the channels.

MalNet-Image Tiny. We construct MALNET-IMAGE TINY, containing 61,201 training, 8,743 validation and 17,486 test images, for *type level classification* experiments by removing the 4 largest types in MALNET-IMAGE. The goal of MALNET-IMAGE TINY is to enable users to rapidly prototype new ideas, since it requires only a fraction of the time needed to train a new model. MALNET-IMAGE TINY is released alongside the full dataset at <https://mal-net.org>.

6.3.2 Interactive Visual Explorer for MALNET-IMAGE

We develop MALNET-IMAGE EXPLORER, an interactive image exploration and visualization tool that enables researchers and practitioners to easily study the data without installa-

tion or download. Figure 6.4 shows MALNET-IMAGE EXPLORER’s desktop web interface and its main components—(1) a hierarchical exploration panel on the left that allows the user to select from the available image types and families; and (2) the image visualization on the right. MALNET-IMAGE EXPLORER’s user interface uses a *responsive* design that automatically adjusts its component layout, based on the users’ device types and screen resolutions. MALNET-IMAGE EXPLORER is available online at: <https://mal-net.org>.

6.4 MalNet-Image Applications

MALNET-IMAGE offers new and unique opportunities to advance the frontiers of cybersecurity research. As examples, we show three exciting new applications made possible by the MALNET-IMAGE database—(1) as a state-of-the-art cybersecurity image benchmark in Section 6.4.1; (2) as the first large-scale public analysis of malicious software detection using binary images in Section 6.4.2; and (3) how to categorize high-risk malware threats (e.g., is this Ransomware or Spyware?) in Section 6.4.3. Then, in Section 6.4.4 we highlight new research directions enabled by MALNET-IMAGE.

Application Setup. We divide MALNET-IMAGE into three stratified sets of data, with a training-validation-test split of 70-10-20 respectively; repeated for both type and family labels (suggested splits available at <https://mal-net.org>). In addition, we conduct malware detection experiments by grouping all 46 malicious software images into one type while the benign type maintains its original label. We evaluate 3 common architectures—ResNet [266], DenseNet [267] and MobileNet [268], based on its macro-F1 score, as is typical for highly imbalanced datasets [194, 214, 261]. Each model is trained for 100 epochs using cross entropy loss (unless specified otherwise) and an Adam optimizer on an Nvidia DGX-1 containing 8 V100 GPUs and 512GB of RAM using Keras with a Tensorflow backend.

6.4.1 Application 1: Benchmarking Techniques

Leveraging the unprecedented scale and diversity of MALNET-IMAGE, we evaluate numerous malware detection and classification techniques that have previously been studied using only private or small-scale databases. Specifically, we evaluate recent techniques including: (a) semantic information encoding via colored channels, (b) model architecture, (c) model pretraining, (d) imbalanced classification techniques, and (e) the performance of MALNET-IMAGE TINY, a small-scale version of MALNET-IMAGE. We detail the setup, results, and analysis of each experiment below.

Table 6.3: We evaluate the performance of 3 popular architectures—ResNet, DenseNet and MobileNetV2—on its macro-F1, macro-precision, and macro-recall. Model performance is similar across architectures, while model size (parameters) and computational cost (MFlops) varies widely. As a result, we conduct all additional experiments using a ResNet18 model as it provides a strong balance between performance and training time.

Model	Params	MFlops	Binary			Type			Family		
			F1	Prec.	Recall	F1	Prec.	Recall	F1	Prec.	Recall
ResNet18	12M	1,820	.86	.89	.84	.47	.56	.42	.45	.54	.42
ResNet50	26M	3,877	.85	.91	.81	.48	.57	.44	.47	.54	.44
ResNet101	45M	7,597	.86	.88	.84	.48	.59	.44	.47	.54	.44
DenseNet121	7.9M	2,872	.86	.90	.83	.47	.56	.43	.46	.53	.44
DenseNet169	14M	3,403	.86	.89	.84	.48	.57	.43	.46	.55	.43
MobileNetV2 (x.5)	1.9M	100	.86	.89	.83	.46	.55	.42	.45	.53	.42
MobileNetV2 (x1)	3.5M	329	.85	.89	.83	.45	.53	.42	.44	.53	.41

Semantic Information Encoding. We evaluate the effect of information encoding in the classification process by training two ResNet18 models—one on the RGB images, where each byte is assigned to a particular color channel depending on its position in the DEX file structure as proposed in [236], and another on grayscale converted images. We find no improvement in the macro-F1 score using semantically encoded RGB images compared to grayscale ones.¹ As there are alternative encoding techniques [236], we believe comparing the effects of different encodings could be an interesting future research direction. Going forward, all models are trained using grayscale images.

Evaluating Model Architectures. We evaluate malware detection and classification performance on 3 popular deep learning architectures—ResNet, DenseNet and MobileNetV2—across a variety of model sizes, using grayscale encoded images, cross entropy loss and no model pretraining. In Table 6.3, we report the macro-F1, macro-precision, and macro-recall of each model. We find that all models obtain similar macro-F1 scores, indicating that a small model has enough capacity to learn the features present in the binary images. Going forward, all experiments use a ResNet18 model due to its strong performance and fast training time.

Transfer learning using ImageNet. We evaluate the effect of pretraining a model using datasets such as ImageNet, and then fine tuning them on binary image data to boost malware classification performance [256, 269, 270, 271, 272]. Following prior work in binary-image model transfer [256, 270, 272], we replace the last layer, freeze all weights

¹Less than 0.001 macro-F1 score differential between grayscale and color performance

except for the last layer, and fine tune it on our training data. We then compare the performance of the pretrained ResNet18 model to one trained from scratch. We find that the pretrained model performs significantly worse than the one trained from scratch, with a macro-F1 of 0.48 versus 0.86, respectively. One potential reason model pretraining is less effective on MALNET-IMAGE, compared to previous work, is due to the large number of images in MALNET-IMAGE’s training set. To put it in perspective, this is $89\times$ more fine tuning data than in [269] and $81\times$ more than in [256].

Accounting for Class Imbalance. Given MALNET-IMAGE’s large class imbalance, we evaluate 3 imbalanced classification techniques: (1) class reweighting with cross entropy loss, (2) focal loss, and (3) class reweighting with focal loss; and compare this to a model trained using cross entropy loss without class weighting. For class reweighting, each example of a class c is weighted according to its effective number $\frac{1-\beta}{1-\beta^{n_c}}$, where n_c is the number of images in class c and $\beta = 0.999$ is selected through a line search across standard values [193] of $\{0.9, 0.99, 0.999, 0.9999\}$. For focal loss [273], a regularization technique that tackles imbalance by establishing margins based on the class size, we set the hyperparameter $\gamma = 2$ as suggested in [273].

Analyzing the results, we find that cross entropy loss with class reweighting improves the *type* macro-F1 score by 0.021, but lowers the binary and family classification scores by 0.002 and 0.006, respectively. In particular, we notice that MALNET-IMAGE’s smallest types benefit the most from class reweighting, where the Click type (113 examples), sees its F1 score rise from 0 to 0.91. On the other hand, focal loss shows no improvement over the baseline model, likely due to its design for use in dense object detectors like R-CNN. Going forward, all experiments use cross entropy loss with class reweighting due to the strong macro-F1 improvement in the smaller malware types.

MALNET-IMAGE TINY Performance. We analyze MALNET-IMAGE TINY by performing type level classification experiments using the optimal model found above—a ResNet18 trained from scratch on grayscale images using cross entropy loss and class reweighting—where the model achieves a macro-F1 score of 0.65, macro-precision of 0.67, and a macro-recall of 0.65. Comparing the results to MALNET-IMAGE, it is unsurprising that macro-F1 score is significantly higher 0.65 vs 0.49, given that the largest 4 types contained a significant proportion of the image diversity (based on the number of families), resulting in an easier classification task.

Limitations. We note that methods that work well on other datasets may not work well on MALNET-IMAGE due to inherent structural differences in the images; vice-versa, methods that work on MALNET-IMAGE may not transfer well to other datasets. We hope

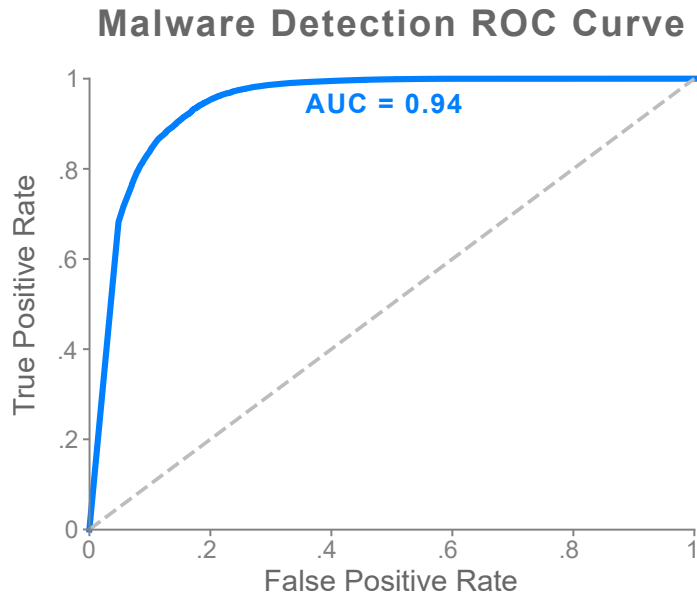


Figure 6.5: Malware detection ROC curve with an AUC of 0.94, demonstrating the potential of binary images as an effective form of malware detection.

MALNET-IMAGE will inspire the release of additional large-scale datasets in the binary image domain, enabling researchers to develop and evaluate new methods that generalize across important domains such as cybersecurity.

6.4.2 Application 2: Malware Detection

Researchers and practitioners can now conduct malware detection experiments on an industry scale dataset, evaluating things that were previously reported in propriety settings. Using the model selected in Section 6.4.1—a ResNet18 model trained from scratch on grayscale images using cross entropy loss and class reweighting—we perform an in-depth analysis of this highly imbalanced detection problem containing 1,182,905 malicious and 79,119 benign images. We find that the model is able to obtain a strong macro-F1 score of 0.86, macro-precision of 0.89 and a macro-recall of 0.84. In Figure 6.5, we further study the model’s detection capabilities by analyzing its ROC curve. The model achieves an AUC score of 0.94, and is able to identify 84% of all malicious files with a false positive rate of 10% (a common threshold used in security [235]). This first of its kind analysis allows researchers insight into malware detection that is usually restricted to handful of industry labs.

Visualizing Model Attention Regions

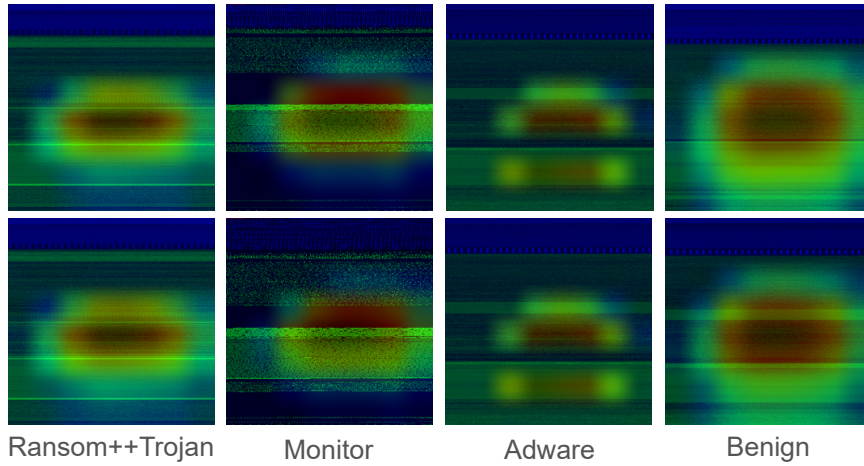


Figure 6.6: Model attention patterns across 4 types of malware (each with 2 images). **Ransom++Trojan:** narrowly focused on thin region of data section. **Benign:** wide range of attention across data section. **Adware:** attention on circular bytecode “hotspots”. **Monitor:** focus on “empty” black region of data section.

6.4.3 Application 3: Malware Classification

MALNET-IMAGE opens up new research into binary images as a tool for multi-class malware classification (e.g., is this file Ransomware or Spyware?). Using the model selected in Section 6.4.1—a ResNet18 model trained from scratch on grayscale images using cross entropy loss and class reweighting—we perform an in-depth analysis of its multi-class classification capability across 47 types and 696 families of malware. We find the model is able to classify the malware *type* and malware *family* with a macro-F1 score of 0.49 and 0.45, respectively. To the best of our knowledge, this is the first time that a large-scale analysis of malware *type* and *family* classification has been conducted, providing a new state-of-the-art benchmark to compare against.

MALNET-IMAGE will be a valuable research tool to support the nascent and promising research direction of analyzing attention maps for malware detection and interpretation. Yakura et al. [274, 265] showed that specific byte sequences found in the attention map closely correlate with malicious code payloads.

To demonstrate the potential of these techniques on MALNET-IMAGE, we use a popular attention mechanism (i.e., Grad-Cam [275]) to highlight regions of interest across 4 types of malware in Figure 6.6. For the Ransom++Trojan and Monitor types (left-side), we can see that model is focused on thin regions of bytecode in the data section; while for the Adware type (middle-right) the model is focused on two separate regions and four circular bytecode “hotspots”. In comparison, attention patterns on Benign images (right-

side) are widely dispersed across the whole data section. One important observation from Figure 6.6 is that model’s attention is focused on the data region of the image (green), as this is where malicious payloads are typically stored. Furthermore, this type of visual analysis significantly reduces the amount of time and effort required to manually investigate a file by guiding security analysts to suspicious regions of the bytecode.

In Figure 6.7, we conduct an in-depth analysis into type level classification performance through a confusion matrix heatmap. A dark diagonal indicates strong classifier performance, where a dark off-diagonal entry indicates poor performance. Each square in the diagonal indicates the percent of examples correctly classified for a particular malware type; and each off-diagonal row entry indicates the percent of incorrectly classified examples for a particular malware type. Four types of malware comprise the majority of misclassifications—Adware, Benign, Riskware and Trojan. Unsurprisingly, these are the 4 largest types of malware (based on the number of images in each class), indicating the strong effect that data imbalance has in the malware classification process. Through the heatmap, we can also identify potential naming disagreements between vendor labels (e.g., “adware” and “adsware”), which can be used as evidence for merging certain types of malware. In addition, we can use the heatmap to view the types of malware the model accurately detects, which is critical in assisting security analysts make informed decisions on high risk threats.

6.4.4 Enabling New Research Directions

The unprecedented scale and diversity of MALNET-IMAGE opens up new exciting research opportunities to the ML and security communities. Below, we present 3 promising directions (R1-R3).

R1. Advancing Vision Based Cybersecurity Research. Research into developing image-based malware detection and classification algorithms has recently surged across industry (e.g., Intel-Microsoft collaboration on Stamina [235], security companies [233, 236]) and academia [237, 238, 239, 240, 241, 249, 250, 251, 252, 253, 254, 255, 256, 247, 257, 258, 245]. However, existing public datasets contain only a handful of classes and thousands of images, and as the field advances, larger and more challenging datasets are needed for the next generation of models. With the release of MALNET-IMAGE, containing over 1.2 million software images across a hierarchy of 47 types and 696 families, researchers now have access to a critical resource to develop and benchmark advanced image-based malware detection and classification algorithms, previously restricted to a few industry labs and research teams.

- R2. Extending Imbalanced Classification into a New and Diverse High-Impact Domain.** While a large body of research has analyzed binary-images in balanced classification settings, only preliminary work has studied malware detection under data imbalance [253] due to the limited number of classes and images available in existing datasets. As a result, it is unknown whether many techniques may generalize to the binary-image domain, and how they will perform in highly imbalanced classification scenarios. We take a first step in studying this by analyzing Figure 6.7, where we can see that classes containing only a few examples typically underperform relative to their more populous counterparts—highlighting the significant challenge of imbalanced classification in the cybersecurity domain. By releasing MALNET-IMAGE, one of the largest naturally imbalanced databases to date, we hope to foster new interest in this important research area, enabling the machine learning community to impact and generalize across domains.
- R3. Interpretable Cybersecurity Research.** Preliminary research has demonstrated the importance of attention mechanisms in binary-image malware classification, where extracted regions can provide strong indicators to human analysts, helping guide them to suspicious parts of the bytecode for analysis [274, 265]. This includes recent research in salience based methods that automatically discover concepts, helping to identify correlated regions of bytecode [276]. Prior to MALNET-IMAGE, researchers were limited to a small number of malicious families and types, hindering their ability to conduct large-scale explainability studies. With MALNET-IMAGE’s nearly 700 classes, researchers can explore a variety of malware, enabling new breakthroughs and discoveries. For example, researchers might discover that new types of visualization and sense-making techniques are needed to accurately summarize large volumes of binary-image data to enhance security analysts decision making capabilities.

6.5 Conclusion

Computer vision research into binary-image malware detection and classification is a crucial tool in protecting enterprise networks and governments from cyber attacks seeking to interrupt critical services and steal intellectual property. Leveraging MALNET-IMAGE’s scale and diversity—containing 1, 262, 024 binary images across a hierarchy of 47 types and 696 families—researchers and practitioners can now conduct experiments that were previously restricted to a few industry labs and research teams. We hope MALNET-IMAGE becomes a central resource for a broad range of research into vision-based cyber defenses, multi-class imbalanced classification, and interpretable security.

Classification Performance: Image Type

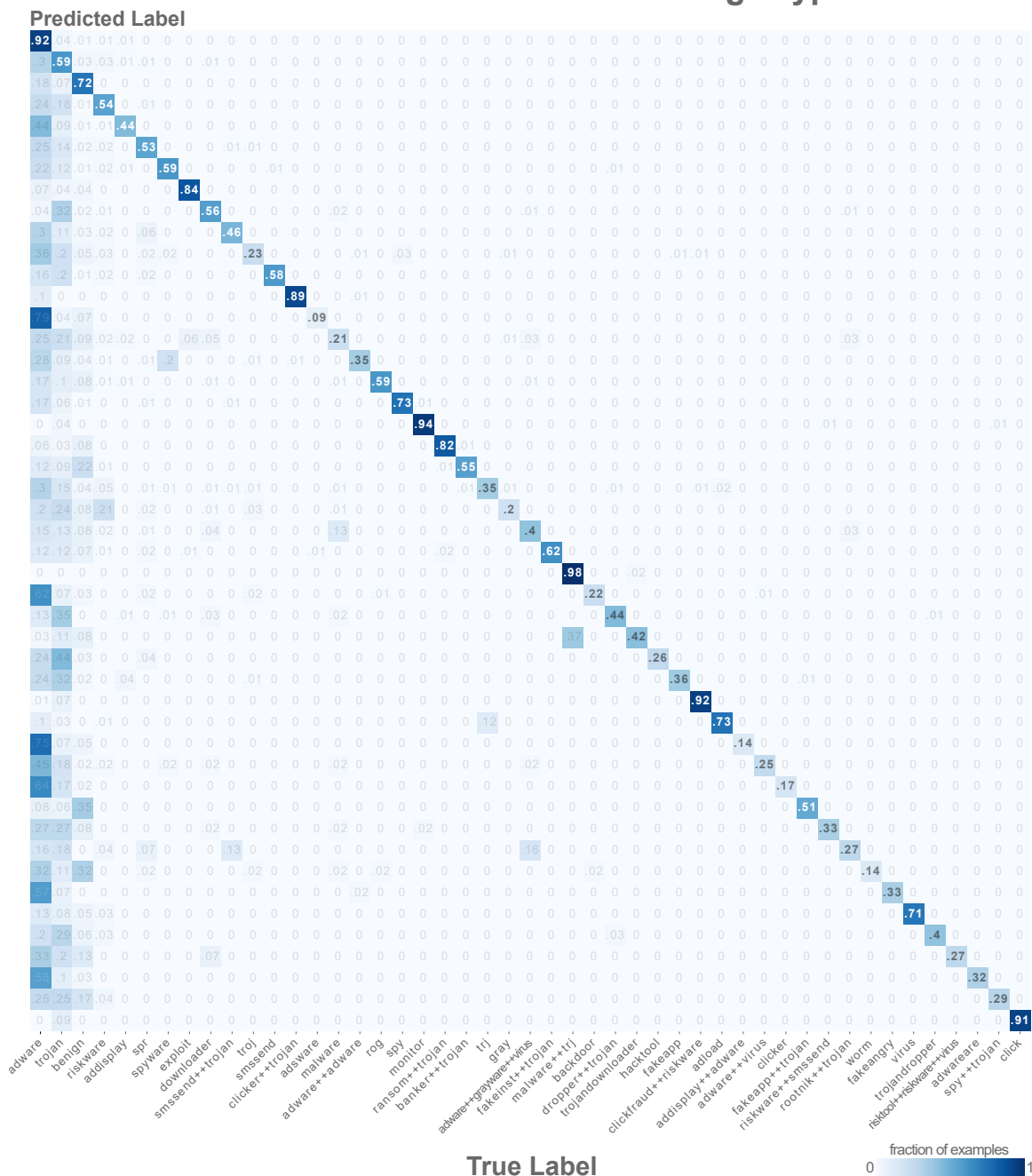


Figure 6.7: Malware classification results using confusion matrix heatmap (classes in descending order of size). We analyze type level classification performance, where a dark diagonal indicates strong performance, and a dark off-diagonal indicates poor performance. Each square in the diagonal indicates the percent of examples correctly classified for a particular malware type; and each off-diagonal entry indicates the percent of incorrectly classified examples for a particular type.

Part IV

Robust Models

Inspired by the success of the deep learning models developed for MALNET-IMAGE in Chapter 6—where our goal was to evaluate state-of-the-art computer vision models for the task of malware classification—and concerned about their fragility to adversarial attack, we develop UNMASK, the first model that flags semantic incoherence in computer vision systems, such as those on self driving cars. UNMASK works by extracting robust features (e.g., beak, wings, eyes) from an image (e.g., “bird”) and comparing them to the expected features of the classification label. For example, if the extracted features for a “bird” image are *wheel*, *saddle* and *frame*, UNMASK alerts that the model may be under attack, and attempts to reclassify the image based on its constituent parts. Our extensive evaluation shows that UNMASK *detects* up to 96.75% of attacks, and *defends* the model by correctly classifying up to 93% of attacks. Clicking on the link below will open its PDF version in the browser:

Chapter 7 UNMASK: Adversarial Detection and Defense Through Robust Feature Alignment. Scott Freitas, Shang-Tse Chen, Zijie J. Wang, Duen Horng Chau. *IEEE International Conference on Big Data (Big Data)*. Online, 2020. <https://arxiv.org/abs/2002.09576>

Inspired by UNMASK’s ability to protect computer visions systems from adversarial attack, we develop REST, which creates noise robust models to reduce undiagnosed sleep disorders through improved sleep monitoring—a fundamental health issue impacting up to 70 million Americans. In the presence of noise, REST improves state-of-the-art sleep stage scoring by 71%—allowing us to diagnose sleep disorders earlier on and in the home environment—while using $19\times$ parameters and $15\times$ less MFLOPS. Clicking on the link below will open its PDF version in the browser:

Chapter 8 REST: Robust and Efficient Neural Networks for Sleep Monitoring in the Wild. Rahul Duggal*, Scott Freitas*, Cao Xiao, Duen Horng Chau, Jimeng Sun. *Proceedings of The Web Conference (WWW)*. Online, 2021.
* Both authors contributed equally to this research <https://arxiv.org/abs/2001.11363>

CHAPTER 7

UNMASK: ADVERSARIAL DETECTION AND DEFENSE THROUGH ROBUST FEATURE ALIGNMENT

Recent research has demonstrated that deep learning architectures are vulnerable to adversarial attacks, highlighting the vital need for defensive techniques to detect and mitigate these attacks before they occur. We present UNMASK, an adversarial detection and defense framework based on robust feature alignment. UNMASK combats adversarial attacks by extracting robust features (e.g., beak, wings, eyes) from an image (e.g., “bird”) and comparing them to the expected features of the classification. For example, if the extracted features for a “bird” image are *wheel*, *saddle* and *frame*, the model may be under attack. UNMASK detects such attacks and defends the model by rectifying the misclassification, re-classifying the image based on its robust features. Our extensive evaluation shows that UNMASK *detects* up to 96.75% of attacks, and *defends* the model by correctly classifying up to 93% of adversarial images produced by the current strongest attack, Projected Gradient Descent, in the gray-box setting. UNMASK provides significantly better protection than adversarial training across 8 attack vectors, averaging 31.18% higher accuracy. We open source the code repository and data: <https://github.com/safreital/unmask>.

7.1 Introduction

In the past few years, deep neural networks (DNNs) have shown significant susceptibility to adversarial perturbation [277, 278]. More recently, a wide range of adversarial attacks [279, 280, 281] have been developed to defeat deep learning systems—in some cases by changing the value of only a few pixels [282]. The ability of these micro perturbations to confuse deep learning architectures highlights a critical issue with modern computer vision systems—that these deep learning systems do not distinguish objects in ways that humans would [283, 284]. For example, when humans see a bicycle, we see its handlebar, frame, wheels, saddle, and pedals (Fig. 7.1, top). Through our visual perception and cognition, we synthesize these detection results with our knowledge to determine that we are actually seeing a bicycle.

However, when an image of a bicycle is adversarially perturbed to fool the model into misclassifying it as a bird (by manipulating pixels), to humans, we still see the bicycle’s **robust features** (e.g., handlebar). On the other hand, the attacked model fails to perceive these robust features, and is tricked into misclassifying the image. How do we incorporate

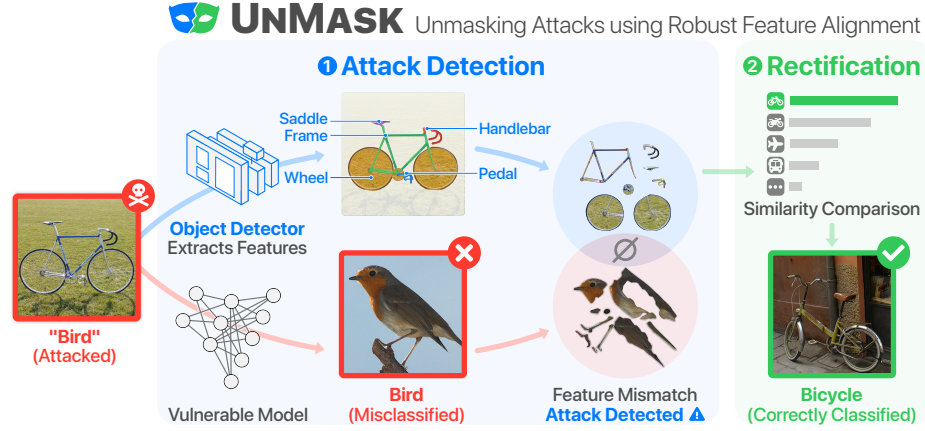


Figure 7.1: UNMASK combats adversarial attacks (in red) through extracting *robust features* from an image (“Bicycle” at top), and comparing them to expected features of the classification (“Bird” at bottom) from the unprotected model. Low feature overlap signals an attack. UNMASK rectifies misclassification using the image’s extracted features. Our approach *detects* 96.75% of gray-box attacks (at 9.66% false positive rate) and *defends* the model by correctly classifying up to 93% of adversarial images crafted by Projected Gradient Descent (PGD).

this intuitive detection capability natural to human beings, into deep learning models to protect them from harm?

It was recently posited that adversarial vulnerability is a consequence of a models’ sensitivity to well-generalizing features in the data [285, 286]. Since models are trained to maximize accuracy, they use *any* available information to achieve this goal. This often results in the use of human incomprehensible features, since a “head” or “wheel” is as natural to a classifier as any other predictive feature. These human incomprehensible (non-robust) features, while useful for improving accuracy, can lead to the creation of adversarially vulnerable models [285]. We extend this notion of adversarial vulnerability as a consequence of non-robust features and develop a framework that protects against attacks by incorporating human priors into the classification pipeline.

7.1.1 Contributions

1. Robust Feature Extraction. We contribute the idea that *robust feature alignment* offers a powerful, explainable and practical method of detecting and defending against adversarial perturbations in deep learning models. A significant advantage of our proposed concept is that while an attacker may be able to manipulate the class label by subtly changing pixel values, it is much more challenging to simultaneously manipulate all the individual features that jointly compose the image. We demonstrate that by adapting an object detector, we can effectively extract higher-level *robust features* contained in images to detect and defend against adversarial perturbations. (Section 7.3.1)

2. UNMASK: Detection & Defense Framework. Building on our core concept of robust feature alignment, we propose UNMASK as a framework to detect and defeat adversarial attacks by quantifying the similarity between the image’s extracted features with the expected features of its predicted class. To illustrate how UNMASK works, we use the example from Figure 7.1, where a bicycle image has been attacked such that it would fool an unprotected model into misclassifying it as a bird. For a real “bird” image, we would expect to see features such as *beak*, *wing* and *tail*. However, UNMASK would (correctly) extract bike features: *wheel*, *frame*, and *pedals*. UNMASK quantifies the similarity between the *extracted* features (of a bike) with the *expected* features (of a bird), in this case zero. This comparison gives us the dual ability to both **detect** adversarial perturbations by selecting a similarity threshold for which we classify an image as adversarial, and to **defend** the model by predicting a corrected class that best matches the extracted features. (Section 7.3.2)

3. Extensive Evaluation. We extensively evaluate UNMASK’s effectiveness using the large UNMASK DATASET that we have newly curated, with over 21k images in total. We test multiple factors, including: 4 strong attacks; 2 attack strength levels; 2 popular CNN architectures; and multiple combinations of varying numbers of classes and feature overlaps. Experiments demonstrate that our approach *detects* up to 96.75% of gray-box attacks with a false positive rate of 9.66% and (2) *defends* the model by correctly classifying up to 93% of adversarial images crafted by Projected Gradient Descent (PGD). UNMASK provides significantly better protection than adversarial training across 8 attack vectors, averaging 31.18% higher accuracy. (Section 7.4)

4. Reproducible Research: Open-source Code & Dataset. We contribute a new dataset incorporating PASCAL-Part [287], PASCAL VOC 2010 [288], a subset of ImageNet [289] and images scraped from Flickr—which we call the UNMASK DATASET. The goal of this dataset is extend the PASCAL-Part and PASCAL VOC 2010 dataset in two ways—(1) by adding 9,236 and 6,592 manually evaluated images from a subset of ImageNet and Flickr, respectively; and (2) by converting PASCAL-Part to the standard Microsoft COCO format [290] for easier use and adoption by the research community. Furthermore, we release this new dataset along with our code and models on GitHub, at <https://github.com/safreital/unmask>.

Throughout the chapter we follow standard notation, using capital bold letters for matrices (e.g., \mathbf{A}), lower-case bold letters for vectors (e.g., \mathbf{a}) and calligraphic font for sets (e.g., \mathcal{S}).

7.2 Background and Related Work

Adversarial attacks typically operate in one of three threat models—(i) white-box, (ii) gray-box or (iii) black-box. In the (i) white-box setting, everything about the model and defense techniques is visible to the attacker, allowing them to tailor attacks to individual neural networks and defense techniques. This is the hardest scenario for the defender since the adversary is aware of every countermeasure. In (ii) the gray-box threat model, we assume that the attacker has access to the classification model but no information on the defense measures. In (iii) the black-box setting, we assume that the attacker has no access to the classification model or the defense techniques. This is the most difficult, and realistic scenario for the attacker since they typically have limited access to the model’s inner workings. Despite this disadvantage, recent research has shown that it’s possible for adversaries to successfully craft perturbations for deep learning models in the black-box setting [291, 292, 293].

7.2.1 Adversarial Attacks

There exists a large body of adversarial attack research. We provide a brief background on the attacks we use to probe the robustness of the UNMASK detection and defense framework. We assume that all attack models operate in a gray-box setting, where the attacker has full knowledge of the classification model, but no knowledge of the defensive measures. We focus on untargeted attacks in all of the experiments.

Projected Gradient Descent (PGD) [294] finds an adversarial example \mathbf{X}_p by iteratively maximizing the loss function $J(\mathbf{X}_p, y)$ for T iterations, where J is the cross-entropy loss.

$$\mathbf{X}_p^{(t+1)} = \mathbf{X}_p^{(t)} + \Pi_\tau \left[\epsilon \cdot \text{sign} \left\{ \nabla_{\mathbf{X}_p^{(t)}} J(\mathbf{X}_p^{(t)}, y) \right\} \right] \quad (7.1)$$

Here $\mathbf{X}_p^{(0)} = \mathbf{X}$ and at every step t , the previous perturbed input $\mathbf{X}_p^{(t-1)}$ is modified with the sign of the gradient of the loss, multiplied by ϵ (attack strength). Π_τ is a function that clips the input at the positions where it exceeds the predefined L_∞ (or L_2) bound τ . We select PGD since it’s one of the strongest first order attacks [294].

MI-FGSM (MIA) [295] is a gradient-based attack utilizing momentum, where it accumulates the gradients \mathbf{g}_t of the first t iterations with a decay factor μ .

$$\mathbf{g}^{(t+1)} = \mu \cdot \mathbf{g}^{(t)} + \frac{\nabla_{\mathbf{X}_p^{(t)}} J(\mathbf{X}_p^{(t)}, y)}{\|\nabla_{\mathbf{X}_p^{(t)}} J(\mathbf{X}_p^{(t)}, y)\|_1} \quad (7.2)$$

$$\mathbf{X}_p^{(t+1)} = \mathbf{X}_p^{(t)} + \Pi_\tau \left[\alpha \cdot \text{sign}(\mathbf{g}^{(t+1)}) \right] \quad (7.3)$$

Here $\alpha = \frac{\epsilon}{T}$, which controls the attack strength. The gradient accumulation (momentum) helps alleviate the trade-off between attack strength and transferability—demonstrated by winning the NIPS 2017 Targeted and Non-Targeted Adversarial Attack competitions.

7.2.2 Adversarial Defense & Detection

Adversarial training seeks to vaccinate deep learning models to adversarial image perturbations by modifying the model’s training process to include examples of attacked images [296, 297]. The idea is that the model will learn to classify these adversarial examples correctly if enough data is seen. It is one of the current state-of-the-art defenses in the white-box setting. When the adversarial examples are crafted by PGD, it is known to improve robustness even against other types of attacks, because PGD is the strongest first-order attack and approximately finds the hardest examples to train [294]. The adversarial training process can be seen in the equation 7.4 below, where we utilize the adversarial perturbations generated by equation 7.1.

$$\min_{\mathbf{W}} \left[\mathbb{E}_{(\mathbf{X}, y) \sim D} \left(\max_{\delta \in \mathcal{S}} L(\mathbf{W}, \mathbf{X} + \delta, y) \right) \right] \quad (7.4)$$

The downside to this technique is that it requires a large amount of data and training time; in addition, it does not incorporate a human prior into the training process.

Alternative Defenses. Defensive distillation is one technique used to robustify deep learning models to adversarial perturbation by training two models—one where the model is trained normally using the provided hard labels and a second model which is trained on soft labels from the probability output of the first model [298]. However, it has been shown that type of technique is likely a kind of gradient masking, making it vulnerable to black-box transfer attacks [299]. In addition, there are many other defensive techniques, some of which include pre-processing the data—which has the goal of eliminating adversarial perturbation before model sees it. A couple of proposed techniques include, image compression [300] and dimensionality reduction [301]. Data pre-processing defense is usually model independent and can easily be used along side with other defenses. The downside of this approach is that most pre-processing techniques have no knowledge of whether the system is actually being attacked. More advanced attacks have also been proposed by replacing the non-differentiable pre-processing step with an approximate differentiable function and back-propagating through the whole pipeline [302, 303].

Adversarial detection attempts to determine whether or not an input is benign or adversarial. This has been studied from multiple perspectives using a variety of techniques, from topological subgraph analysis [304] to image processing [305, 306, 307], and hidden/output layer activation distribution information [308, 309, 310].

7.3 UNMASK: Detection and Defense Framework

UNMASK is a new method for protecting deep learning models from adversarial attacks by using *robust features* that semantically align with human intuition to combat adversarial perturbations (Figure 7.1). The objective is to defend a **vulnerable** deep learning model M (Figure 7.1, bottom) using our **UNMASK** defense framework D (Figure 7.1, top), where the adversary has full access to M but is unaware of the defense strategy D , constituting a *gray-box* attack on the overall classification pipeline [300]. In developing the UNMASK framework for adversarial detection and defense, we address three sub-problems:

1. **Identify robust features** by training a model K that takes as input, input space \mathbf{X} and maps it to a set of robust features $\mathcal{F} = \{K : \mathbf{X} \rightarrow \mathbb{R}\}$.
2. **Detect** if input space \mathbf{X} is benign (+1) or adversarially perturbed (-1) by creating a binary classifier $C : \mathcal{F} \rightarrow \{\pm 1\}$ that utilizes robust features \mathcal{F} .
3. **Defend** against adversarial attacks by developing a classifier $C : \mathcal{F} \rightarrow y$ to predict y using robust features \mathcal{F} .

We present our solution to problem 1 in Section 7.3.1; and discuss how to solve problems 2 and 3 in Section 7.3.2.

7.3.1 Aligning Robust Features with Human Intuition

In order to discuss adversarially robust features, we categorize features into three distinct types: *p-useful*, (ii) *γ -robust*, and (iii) **useful but non-robust** [285]. For a distribution D , a feature is defined as *p-useful* ($p > 0$) if it is correlated with the true label in expectation (Eq. 7.5). These *p-useful* features are important for obtaining high accuracy classifiers.

$$\mathbb{E}_{(\mathbf{X}, y) \sim D} [y \cdot f(\mathbf{X})] \geq p \quad (7.5)$$

A feature is referred to as a *robust feature* (γ -robust) if that feature remains useful ($\gamma > 0$) under a set of allowable adversarial perturbations $\delta \in S$. This is defined in

Equation 7.6.

$$\mathbb{E}_{(\mathbf{X}, y) \sim D} \left[\inf_{\delta \in S} y \cdot f(\mathbf{X} + \delta) \right] \geq \gamma \quad (7.6)$$

Lastly, a useful but non-robust feature is a feature that is p -useful for some $p > 0$ but not γ -robust for any $\gamma \geq 0$. These non-robust features are useful for obtaining high accuracy classifiers, but can be detrimental in the presence of an adversary since they are often weakly correlated with the label and can be easily perturbed.

When training a classifier, minimizing classification loss makes no distinction between robust and non-robust features—it only distinguishes whether a feature is p -useful. This causes a model to utilize useful but non-robust features in order to improve classifier accuracy. However, in the presence of adversarial perturbation, these useful but non-robust features become anti-correlated with the label and lead to misclassification. On the other hand, if a model is trained from purely *robust features*, it has lower classification performance; but gains non-trivial adversarial robustness [285].

Our goal is to enable a model M to use both γ -robust and useful but non-robust features to achieve the highest possible classification accuracy, while utilizing only the γ -robust features to determine if an image is attacked. This allows model M to use *any* signal in the data to improve classification accuracy, while the defense framework uses *only* the robust features to provide a safeguard against adversarial perturbations. In order to determine the γ -robust features from an input space \mathbf{X} , we develop model K to identify robust features by training on a robust distribution \hat{D}_R that satisfies Equation 7.7.

$$\mathbb{E}_{(\mathbf{X}, y) \sim \hat{D}_R} [f(\mathbf{X}) \cdot y] = \begin{cases} \mathbb{E}_{(\mathbf{X}, y) \sim D} [f(\mathbf{X}) \cdot y] & \text{if } f \in \mathcal{F} \\ 0, & \text{otherwise} \end{cases} \quad (7.7)$$

Here \mathcal{F} represents the set of human-level robust features identified in the Pascal-Part dataset using segmentation masks. Ideally, we want these human-level features to be as useful as the original distribution D , while excluding the useful but non-robust features. Using robust dataset \hat{D}_R , we train a robust feature extraction model K with weights \mathbf{W} to recognize *only* the human-level robust features using Equation 7.8.

$$\min_{\mathbf{W}} \left[\mathbb{E}_{(\mathbf{X}, y) \sim \hat{D}_R} L(\mathbf{X}, y) \right] \quad (7.8)$$

From a practical standpoint, we adopt a Mask R-CNN architecture [311] for feature extraction model K . We considered multiple approaches, but decided to use Mask R-CNN for its ability to leverage image segmentation masks to learn and identify coherent image

regions that closely resemble robust features that would appear semantically and visually meaningful to humans. Different from conventional image classification models or object detectors, the annotations used to train our robust feature extractor K are *segmented* object parts instead of the whole objects. For example, for the *wheel* feature, an instance of training data would consist of a bike image and a *segmentation mask* indicating which region of that image represents a wheel. Technically, this means K uses only a part of an image, and not the whole image, for training (See Table 7.1). Furthermore, while an image may consist of multiple image parts, K treats them independently.

7.3.2 Robust Features For Detection and Defense

Leveraging the robust features extracted from model K , we introduce UNMASK as a detection and defense framework (D). For an *unprotected* model M (Figure 7.1, bottom), an adversary crafts an *attacked* image by carefully manipulating its pixel values using an adversarial technique (e.g., PGD [294]). This attacked image then fools model M into misclassifying the image, as shown in Figure 7.1. To guard against this kind of attack on M , we use our UNMASK defense framework D in conjunction with the *robust feature extraction model* K (Figure 7.1, top).

Model K processes the same image, which may be benign or attacked, and extracts the robust features from the image to compare to the images' expected features. Figure 7.1 shows an example, where an attacked *bike* image has fooled the unprotected model M to classify it as a *bird*. We would *expect* the robust features to include *head*, *claw*, *wing*, and *tail*. However, from the same (attacked) image, UNMASK's model K extracts *wheels*, *handle* and *seat*. Comparing the set of *expected* features and the actual *extracted* features (which do not overlap in this example), UNMASK determines the image was attacked, and predicts its class to be *bike* based on the extracted features. This robust feature alignment forges a layer of protection around a model by disrupting the traditional pixel-centric attack [279, 294, 282]. This forces the adversary to solve a more complex problem of manipulating both the class label and all of the image's constituent parts. For example, in Figure 7.1 the attacker needs to fool the defensive layer into misclassifying the bike as a bird by, (1) changing the class label and (2) manipulating the robust bike features (*wheel*, *seat*, *handlebar*) into bird features. UNMASK's technical operations for detection and defense are detailed below and in Algorithm 3:

1. **Classify** input space \mathbf{X} to obtain prediction \hat{y} from unprotected model M , i.e., $\hat{y} = M(\mathbf{X})$. At this point, UNMASK does not know if \mathbf{X} is adversarial or not.
2. **Extract robust features** of \mathbf{X} using robust feature extraction model K , i.e., $f_r =$

Algorithm 3: UNMASK

Input: Data distribution D , unprotected model M , class-feature matrix \mathbf{V} , input space \mathbf{X}_t , threshold t
Result: adversarial prediction $z \in \{-1, 1\}$, predicted class p

1 $\mathbb{E}_{(\mathbf{X}, y) \sim \hat{D}_R} [f(\mathbf{X}) \cdot y] = \begin{cases} \mathbb{E}_{(\mathbf{X}, y) \sim D} [f(\mathbf{X}) \cdot y] & \text{if } f \in \mathcal{F} \\ 0, & \text{otherwise} \end{cases} \quad (\text{create } \hat{D}_R)$

2 $K = \min_{\mathbf{W}} \left[\mathbb{E}_{(\mathbf{X}, y) \sim \hat{D}_R} L(\mathbf{X}, y) \right];$

3 $f_r = K(\mathbf{X}_t); \quad (\text{extracted features})$

4 $f_e = \mathbf{V}[M(\mathbf{X}_t)]; \quad (\text{expected features})$

5 **Detection:**

6 $s = JS(f_r, f_e); d = 1 - s; \quad (\text{JS = Jaccard similarity})$

7

$$z = \begin{cases} +1 \text{ (benign)}, & \text{if } d < t \\ -1 \text{ (adversarial)}, & \text{if } d \geq t \end{cases}$$

8 **Defense:**

9

$$p = \begin{cases} \hat{y}, & \text{if } z = +1 \\ \underset{c \in C}{\operatorname{argmin}} JS(f_e, \mathbf{V}[c]), & \text{if } z = -1 \end{cases}$$

10 **return** $z, p;$

$K(\mathbf{X})$, where $f_r \subseteq \mathcal{F}$. Armed with these features f_r , UNMASK detects if model M is under attack, and rectifies misclassification.

3. **Detect attack** by measuring the similarity between the *extracted* features f_r and the set of *expected* features $f_e = \mathbf{V}[\hat{y}]$ —where \mathbf{V} is the feature matrix in Table 7.1—by calculating the Jaccard similarity score $s = JS(f_e, f_a)$. If distance score $d = 1 - s$ is greater than threshold t , input \mathbf{X} is deemed benign, otherwise adversarial. Adjusting t allows us to assess the trade-off between sensitivity and specificity, which we describe in detail in Section 7.4.
4. **Defend and rectify** an input to be adversarial also means that model M is under attack and is giving unreliable classification output. Thus, we need to rectify the misclassification. UNMASK accomplishes this by comparing the robust extracted features f_r to every set of class features in \mathbf{V} , outputting class \hat{y} that contains the highest feature similarity score s , where $0 \leq s \leq 1$.

7.4 Evaluation

We extensively evaluate UNMASK’s effectiveness in **defending** and **detecting** adversarial perturbations using: 4 strong attacks across two strength levels; 2 popular CNN architectures as unprotected models M ; multiple combinations of varying numbers of classes and feature overlaps; and benchmarking UNMASK against one of the strongest adversarial defenses—adversarial training [294]. All experiments are conducted in a Linux environment on a DGX-1 running Python 3.6 with open-source libraries Keras, Tensorflow, PyTorch, Advertorch [312] and Matterport [313];

7.4.1 Experiment Setup

Experiments are conducted in a Linux environment using Python 3 on an Nvidia DGX-1. We open-source all of the code, data and models used in this work. For additional information on using UNMASK, we provide a detailed walk through at <https://github.com/safreital/unmask>.

UnMask dataset. We curated the UNMASK DATASET for evaluation, which consists of four component datasets—PASCAL-Part, PASCAL VOC 2010, a subset of ImageNet and images scraped from Flickr (Table 7.2). To ensure the Flickr images are not duplicates, we compare the perceptual hash of each Flickr image to ImageNet and VOC’10. The goal of this curation is to (i) collect all the data used in our evaluation as a single source for use by the research community, and (ii) to increase the number of images available for evaluating the performance of the deep learning models and the UNMASK defense framework. We designed multiple *class sets* with varying number of classes and feature overlap (e.g., CS3a, in Table 7.4; and Table 7.1), to study how they affect detection and defense effectiveness. We call each combination of *class count* and *feature overlap* a “class set”, abbreviated as “CS.” CS3 thus means a class set with 3 classes. CS3a and CS3b have the same number of classes, with different feature overlap. We further discuss the utilization of data below.

In Table 7.1, the class-feature matrix describes the features contained by each class in the dataset. The PASCAL-Part dataset has 18 variations of the leg feature, however, in order to create a model that better generalizes, we combine this to a single leg feature. We note that in Table 7.1, that two features have multiple sub-features condensed into a single feature (not listed due to space constraints). These features are: vehicle: {vehicle left, vehicle right, vehicle top, vehicle back} and coach: {coach left, coach right, coach back, coach top, coach front}. In addition, we note that there is a minor error in the conversion of the handlebar feature in the bike and motorcycle class (handlebar features were labeled as hand). However, since those classes are not utilized in the experiments, the effects are

minimized.

Adversarial attacks. We evaluate UNMASK on 4 attacks:

- **PGD- L_∞** , one of the strongest first-order attacks [294]. Its key parameter ϵ represents the allowed per-pixel perturbation. For example, $\epsilon = 4$ means changing up to 4 units of intensity (out of 255). It is common to evaluate ϵ up to 16, with a stepsize of 2 and 20 iterations [300, 296, 294].
- **PGD- L_2** we also evaluate PGD in the L_2 norm, which bounds the ϵ perturbation across the whole image, instead of a per-pixel bound as in the L_∞ norm. Since the perturbation is bounded across the whole image, ϵ is naturally larger—typically ranging from 300-900 [314].
- **MI-FGSM L_∞ (MIA- L_∞)** is a strong gradient attack with key parameters μ and ϵ (see PGD- L_∞). We set decay factor $\mu=1$ as it provides the most effective attack [295].
- **MI-FGSM L_2 (MIA- L_2)** we also evaluate MI-FGSM in the L_2 norm, bounding ϵ perturbation across the whole image, instead of per-pixel as in L_∞ (see PGD- L_2).

Adversarial defense. We compare against adversarial training, one of the strongest ad-

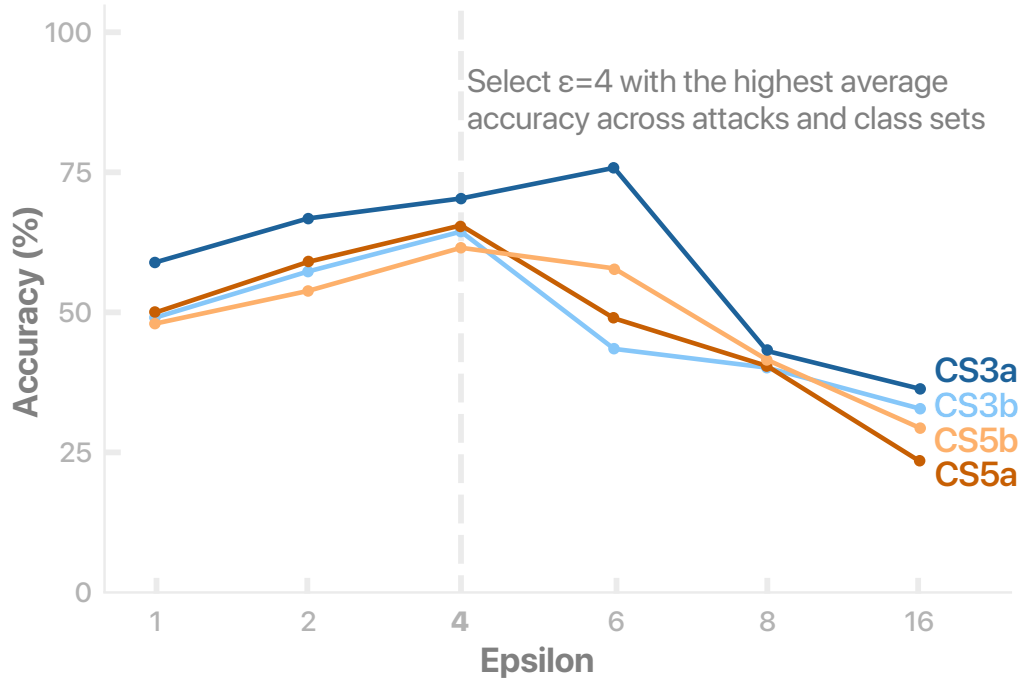


Figure 7.2: Line search for adversarial training parameter ϵ on validation data. We select $\epsilon = 4$, since it provides the best performance on most attacks.

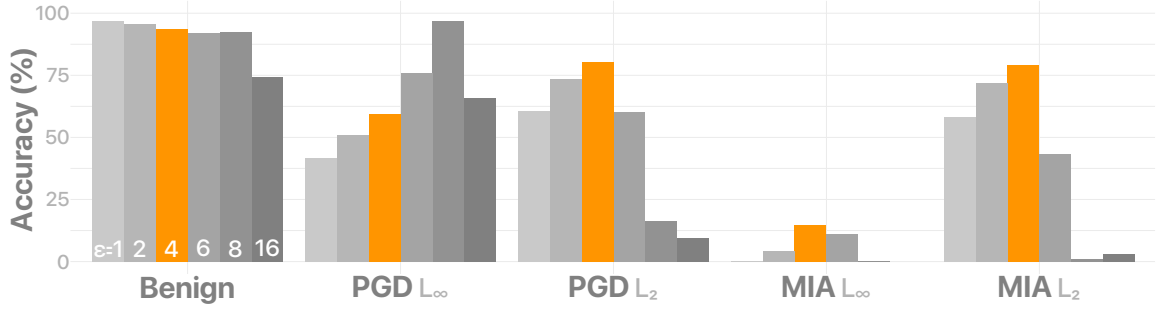


Figure 7.3: Detailed bar chart describing the performance of ϵ across all attack vectors. We select $\epsilon = 4$, since it provides the best performance on most attacks.

versarial defense techniques. To select ϵ —which controls the perturbation strength of adversarial training—we follow standard procedure [294] and determine ϵ on a per-dataset basis (class set). Correctly setting this parameter is critical since a small ϵ value will have no effect on robustness, while too high a value will lead to poor benign accuracy. Following standard procedure, we select ϵ on a per-dataset basis (class set in our case) [294] by conducting a line search across $\epsilon = \{1, 2, 4, 6, 8, 16\}$. We find that $\epsilon = 4$, provides the best performance on the validation set across each class sets, as seen in Figure 7.2. While $\epsilon = 6$ appears to be a good choice for class set CS3a, it has poor generalization performance and overfits to PGD- L_∞ . This can be seen through the bar chart of Figure 7.3, where $\epsilon > 4$ reduces the generalization of adversarial training to all other attack vectors. For this reason, we select $\epsilon = 4$ for all class sets.

Training robust feature extraction model. As illustrated in Figure 7.1, the robust feature extraction model K takes an image as input (e.g., bike) and outputs a set of features (e.g., wheel,...). To train K , we use the PASCAL-Part dataset [287], which consists of 180,423 feature segmentation masks over 9,323 images across the 44 robust features. The original dataset contains very fine-grained features, such as 18 types of “legs” (e.g., right front lower leg, left back upper leg), while for our purposes we only need the abstraction of “leg”. Therefore, we combined these fine-grained features into more generalized ones (shown as rows in Table 7.1).

We train K for 40 epochs, following a similar procedure described in [313]. We use a ratio of 80/10/10 for training, validating and testing the model respectively (see Table 7.2). Our work is the first adaptation of Mask R-CNN model for the PASCAL-Part dataset. As such, there are no prior results for comparison. We computed model K ’s mean Average Precision (mAP), which estimates K ’s ability to extract features. The model attains an mAP of 0.56, in line with Mask R-CNN on other datasets [311]. Model K processes up to 4 images per second with a single Nvidia Titan X, matching the speeds reported in

[313]. This speed can be easily raised through parallelism by using more GPUs. As robust feature extraction is the most time-intensive process of the UNMASK framework, its speed is representative of the overall speed of the framework.

Training the unprotected model. As described in Section 7.3, M is the model under attack, and is what UNMASK aims to protect. In practice, the choice of architecture for M and the data it is trained on are determined by the application. Here, our evaluation studies two popular deep learning architectures — ResNet50 [315] and DenseNet121 [267]— however, UNMASK supports other architectures. Training these models from scratch is generally computationally expensive and requires large amount of data. To reduce such need for computation and data, we adopt the approach described in [313], where we leverage a model pre-trained on ImageNet images, and *replace* its dense layers (i.e., the fully connected layers) to enable us to work with various class sets (e.g., CS3a). Refer to Table 7.2, for a breakdown of the data used for training and evaluation.

7.4.2 Evaluating UnMask Defense and Detection

The key research questions that our evaluation aims to address is how effective UNMASK can (1) **detect** adversarial images, and (2) **defend** against attacks by rectifying misclassification through inferring the actual class label. We scrape Flickr (see Table 7.2) to obtain a large number of unseen images matching our class sets. We note that evaluation is focused on images containing a single-class (i.e., no “person” and “car” in same image) as this allows for a more controlled environment.

Evaluating defense and rectification. As the defense evaluation focus is on rectifying misclassification, our test images have a contamination level of 1—meaning all of the images are adversarial. Comparing UNMASK to adversarial training (AT), we find that utilizing robust features that semantically align with human intuition provides a significant improvement over γ -robust features learned through adversarial training. We begin with a high-level analysis in Figure 7.4, comparing UNMASK to adversarial training (“AT”) and no defense (“None”). UNMASK’s robust feature alignment performs 31.18% better than adversarial training and 74.44% than no defense when averaged across 8 attack vectors and all class sets (see Figure 7.4).

In Table 7.4, we analyze the information contained in Figure 7.4 in detail. One key observation is that *feature overlap* is a dominant factor in determining the accuracy of the UNMASK defense, as opposed to the number of classes. When examining the ResNet50 model on MIA- L_∞ ($\epsilon=8$), class set CS3b (3 classes; feature overlap 50%), UNMASK is able to determine the underlying class 77% of the time. At class set CS5a (5 classes;

feature overlap 23.53%) an accuracy of 79% is obtained, highlighting the important role that feature overlap plays in UNMASK’s defense ability. Similar trends can be observed across many of the attacks. In addition, Table 7.4 highlights that UNMASK is agnostic to the deep learning model that is being protected (ResNet50 vs DenseNet121), as measured by the ability of UNMASK to infer an adversarial images’ actual class.

It is interesting to observe that $MIA-L_\infty$ is more effective at breaking the UNMASK defense. We believe this could be due to the single-step attacks’ better transferability, which has been reported in prior work [296]. We also note the fact that UNMASK’s accuracy could be higher than the un-attacked model M if model K learns a better representation of the data through the feature masks as opposed to model M , which trains on the images directly.

Evaluating attack detection. To evaluate UNMASK’s effectiveness in detecting adversarial images, we set the contamination level to 0.5—meaning half of the images are benign and the other half are adversarial. Figure 7.5 summarizes UNMASK’s detection effectiveness, using *receiver operating characteristics* (ROC) curves constructed by varying the adversarial-benign threshold t . The curves show UNMASK’s performance across operating points as measured by the tradeoff between *true positive* (TP) and *false positive* (FP) rates. Table 7.3 shows the number of images used to test the detection ability of UNMASK. Only images that are successfully attacked are used for evaluation (combined with benign counterparts), thus the variations in numbers.

An interesting characteristic of UNMASK’s protection is that its effectiveness may not be affected strictly based on the number of classes in the dataset as in conventional classification tasks. Rather, an important factor is how much *feature overlap* there is among the classes. The ROC curves in Figure 7.5 illustrate this phenomenon, where UNMASK provides better detection when there are 5 classes (Figure 7.5, dark orange) than when there are 3 classes (light blue). As shown in Table 7.2, the 5-class setup (CS5a—dark orange) has

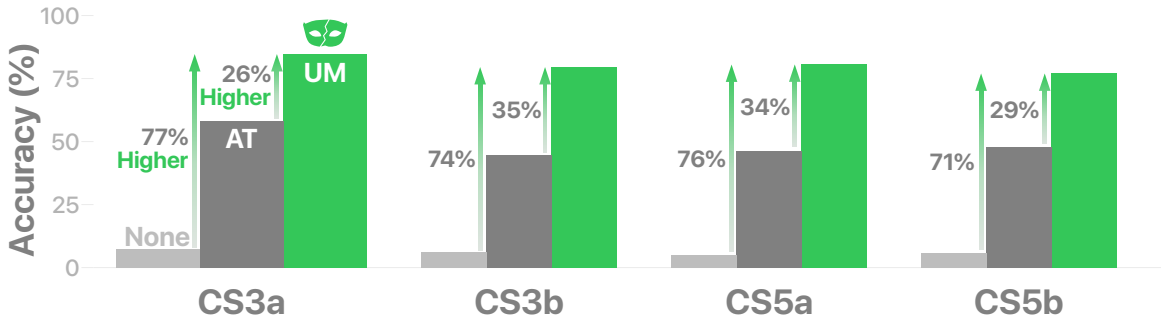


Figure 7.4: Accuracies (in %) for each class set averaged across all attack vectors, strengths, and models from Table 7.4. On average, UNMASK (UM) performs 31.18% better than adversarial training (AT) and 74.44% than no defense (None).

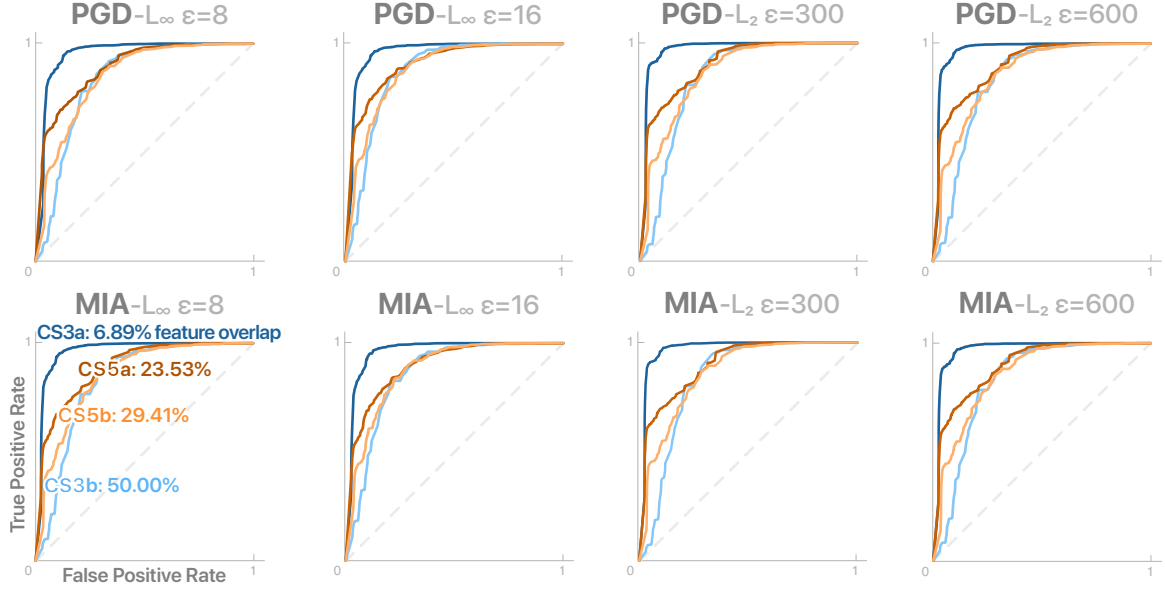


Figure 7.5: UNMASK’s effectiveness in detecting 4 strong attacks at two strength levels. UNMASK’s protection may not be affected strictly based on the number of classes. Rather, an important factor is the *feature overlap* among classes. UNMASK provides better detection when there are 5 classes (dark orange; 23.53% overlap) than when there are 3 (light blue; 50% overlap). Keeping the number of classes constant and varying their feature overlap also supports our observation about the role of feature overlap (e.g., CS3a at 6.89% vs. CS3b at 50%). Dotted line indicates random guessing.

a feature overlap of 23.53% across the the 5 classes’ 34 unique features, while the 3-class setup (CS3b—light blue) has 50% overlap. Keeping the number of classes constant and varying their feature overlap also supports this observation about the role of feature overlap (e.g., CS3a vs. CS3b in Figure 7.5).

For a given feature overlap level, UNMASK performs similarly across attack methods. When examining feature overlap 6.89% (CS3a) on DenseNet121, UNMASK attains AUC scores of 0.95, 0.958, 0.968, 0.967, 0.962, 0.961, 0.969 and 0.967 on attacks PGD- L_∞ ($\epsilon=8/16$), PGD- L_2 ($\epsilon=300/600$), MIA- L_∞ ($\epsilon=8/16$) and MIA- L_2 ($\epsilon=300/600$), respectively. This result is significant because it highlights the ability of UNMASK to operate against multiple strong attack strategies to achieve high detection success rate. As a representative ROC operating point for the attack vectors, we use MIA- L_2 ($\epsilon=300$) on feature overlap 6.89%. In this scenario, UNMASK is able to detect up to 96.75% of attacks with a false positive rate of 9.66%. We believe that performing well in a low feature overlap environment is all that is required. This is because in many circumstances it is not important to distinguish the exact class (e.g., dog or cat) of the image, but whether the image is being completely misclassified (e.g., car vs. person). Therefore, in practice, classes can be selected such that feature overlap is minimized.

7.5 Conclusion

In this chapter, we have introduced a new method for semantically aligning robust features with human intuition, and showed how it protects deep learning models against adversarial attacks through the UNMASK detection and defense framework. Through extensive evaluation, we analyze the merits of UNMASK’s ability to *detect* attacks—finding up to 96.75% of attacks with a false positive rate of 9.66%; and *defend* deep learning models—correctly classifying up to 93% of adversarial images in the gray-box scenario. UNMASK provides significantly better protection than adversarial training across 8 attack vectors, averaging 31.18% higher accuracy. Our proposed method is fast and architecture-agnostic. We expect our approach to be one of multiple techniques used in concert to provide comprehensive protection. Fortunately, our proposed technique can be readily integrated with many existing techniques, as it operates in parallel to the deep learning model that it aims to protect.

Table 7.1: Class-Feature Matrix. Top: dots mark classes’ features. Bottom: four class sets with varying levels of feature overlap. Features *vehicle* and *coach* have sub-features not listed here due to space (see Github repository).

Features	Airplane	Bicycle	Bird	Boat	Bottle	Bus	Car	Cat	Chair	Cow	Dining Table	Dog	Horse	Motorbike	Person	Potted Plant	Sheep	Sofa	Train	Television
Arm																				
Beak																				
Body																				
Cap																				
Coach																				
Door																				
Engine																				
Ear																				
Eye																				
Eyebrow																				
Foot																				
Front side																				
Hair																				
Hand																				
Head																				
Headlight																				
Hoof																				
Horn																				
Leg																				
License plate																				
Mirror																				
Mouth																				
Muzzle																				
Neck																				
Nose																				
Paw																				
Plant																				
Pot																				
Saddle																				
Screen																				
Stern																				
Tail																				
Torso																				
Vehicle																				
Wheel																				
Window																				
Wing																				
Class Set																				
CS3a																				
CS3b																				
CS5a																				
CS5b																				

Table 7.2: Number of images used to train and evaluate models K , M and defense framework D . We train K on PASCAL-Part dataset, and model M on PASCAL VOC 2010 plus a subset of ImageNet. Four *class sets* are investigated in the evaluation, with varying classes and feature overlap. We evaluate model M and defense framework D on Flickr.

Experimental Setup					PASCAL-Part			VOC+Net	Flickr		
Model	Class set	Classes	Parts	Overlap	Train	Val	Test	Train	Val	Test	
K	-	44	-	-	7,457	930	936	-	-	-	
	CS3a	3	29	6.89%	-	-	-	7,780	1,099	2,351	
M	CS3b	3	18	50.00%	-	-	-	9,599	1,339	2,867	
	CS5a	5	34	23.53%	-	-	-	11,639	1,477	3,179	
	CS5b	5	34	29.41%	-	-	-	13,011	1,928	4,129	

Table 7.3: Number of images used to evaluate the detection capability of UNMASK. Only images that are successfully attacked are used for evaluation (combined with their benign counterparts), thus the variations in numbers. We report values for PGD and MIA with $\epsilon=16$, respectively. Numbers are similar for $\epsilon=8$.

Class Set	Number of Images Evaluated			
	PGD- L_∞	PGD- L_2	MIA- L_∞	MIA- L_2
CS3a	3,648	4,702	4,702	4,702
CS3b	4,652	5,732	5,734	5,734
CS5a	5,412	6,358	6,358	6,358
CS5b	6,822	8,256	8,258	8,258

Table 7.4: Accuracies in countering 4 strong attacks at 2 strength levels (PGD- L_∞ , PGD- L_2 , MIA- L_∞ , MIA- L_2), using 2 CNN architectures as unprotected model M across 4 class sets. UNMASK (“UM”) provides significantly better protection than adversarial training (“AT”), 31.18% on average. “None” means no defense.

Setup		No Attk			PGD- L_∞ $\epsilon = 8$			PGD- L_∞ $\epsilon = 16$			PGD- L_2 $\epsilon = 300$			PGD- L_2 $\epsilon = 600$		
M	CS	None	AT	UM	None	AT	UM	None	AT	UM	None	AT	UM	None	AT	UM
ResNet	3a	.98	.96	.94	.31	.71	.85	.22	.50	.72	.07	.86	.92	.00	.67	.91
	3b	.97	.94	.92	.24	.63	.82	.19	.47	.68	.01	.75	.89	.00	.31	.85
	5a	.97	.92	.93	.17	.51	.82	.15	.24	.66	.00	.79	.91	.00	.57	.89
	5b	.97	.92	.91	.22	.56	.78	.17	.34	.61	.04	.78	.88	.00	.50	.84
DenseNet	3a	.97	.95	.94	.31	.70	.86	.24	.48	.74	.02	.86	.93	.00	.71	.91
	3b	.97	.93	.92	.25	.60	.82	.23	.44	.67	.02	.79	.89	.00	.46	.85
	5a	.97	.90	.93	.22	.51	.82	.18	.27	.66	.03	.77	.91	.00	.54	.88
	5b	.97	.92	.91	.24	.55	.79	.21	.28	.62	.02	.81	.89	.00	.58	.85
<hr/>																
Setup		No Attk			MIA- L_∞ $\epsilon = 8$			MIA- L_∞ $\epsilon = 16$			MIA- L_2 $\epsilon = 300$			MIA- L_2 $\epsilon = 600$		
M	CS	None	AT	UM	None	AT	UM	None	AT	UM	None	AT	UM	None	AT	UM
ResNet	3a	.98	.96	.94	.00	.22	.82	.00	.00	.68	.01	.86	.92	.00	.68	.91
	3b	.97	.94	.92	.00	.02	.77	.00	.00	.63	.00	.68	.89	.00	.29	.85
	5a	.97	.92	.93	.00	.25	.79	.00	.02	.63	.00	.79	.91	.00	.58	.89
	5b	.97	.92	.91	.00	.12	.73	.00	.01	.55	.01	.77	.87	.00	.51	.84
DenseNet	3a	.97	.95	.94	.00	.37	.83	.00	.02	.69	.00	.86	.92	.00	.72	.90
	3b	.97	.93	.92	.00	.18	.76	.00	.01	.62	.00	.78	.89	.00	.49	.85
	5a	.97	.90	.93	.00	.27	.78	.00	.02	.58	.00	.77	.91	.00	.56	.87
	5b	.97	.92	.91	.00	.30	.75	.00	.03	.58	.00	.80	.89	.00	.60	.84

CHAPTER 8

REST: ROBUST AND EFFICIENT NEURAL NETWORKS FOR SLEEP MONITORING IN THE WILD

In recent years, significant attention has been devoted towards integrating deep learning technologies in the healthcare domain. However, to safely and practically deploy deep learning models for home health monitoring, two significant challenges must be addressed: the models should be (1) robust against noise; and (2) compact and energy-efficient. We propose REST, a new method that simultaneously tackles both issues via 1) *adversarial training* and controlling the Lipschitz constant of the neural network through *spectral regularization* while 2) enabling neural network compression through *sparsity regularization*. We demonstrate that REST produces highly-robust and efficient models that substantially outperform the original full-sized models in the presence of noise. For the sleep staging task over single-channel electroencephalogram (EEG), the REST model achieves a macro-F1 score of 0.67 vs. 0.39 achieved by a state-of-the-art model in the presence of Gaussian noise while obtaining $19\times$ parameter reduction and $15\times$ MFLOPS reduction on two large, real-world EEG datasets. By deploying these models to an Android application on a smartphone, we quantitatively observe that REST allows models to achieve up to $17\times$ energy reduction and $9\times$ faster inference. We open source the code repository <https://github.com/duggalrahul/REST>.

8.1 Introduction

As many as 70 million Americans suffer from sleep disorders that affects their daily functioning, long-term health and longevity. The long-term effects of sleep deprivation and sleep disorders include an increased risk of hypertension, diabetes, obesity, depression, heart attack, and stroke [19]. The cost of undiagnosed sleep apnea alone is estimated to exceed 100 billion in the US [20].

A central tool in identifying sleep disorders is the **hypnogram**—which documents the progression of sleep stages (**REM** stage, **Non-REM** stages **N1** to **N3**, and **Wake** stage) over an entire night (see Fig. 8.1, top). The process of acquiring a hypnogram from raw sensor data is called **sleep staging**, which is the focus of this work. Traditionally, to reliably obtain a hypnogram the patient has to undergo an overnight sleep study—called *polysomnography* (PSG)—at a sleep lab while wearing bio-sensors that measure physiological signals, which include electroencephalogram (EEG), eye movements (EOG), muscle activity or skeletal

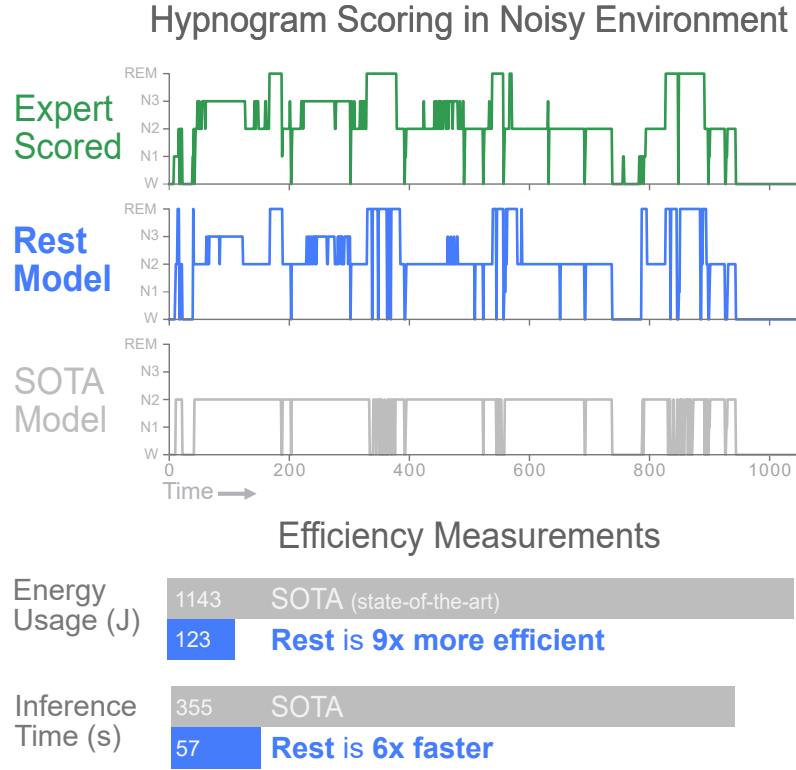


Figure 8.1: Top: we generate hypnograms for a patient in the SHHS test set. In the presence of Gaussian noise, our **REST**-generated hypnogram closely matches the contours of the **expert**-scored hypnogram. Hypnogram generated by a state-of-the-art (**SOTA**) model by Sors et al. [316] is considerably worse. Bottom: we measure energy consumed (in Joules) and inference time (in seconds) on a smartphone to score one night of EEG recordings. REST is 9X more energy efficient and 6X faster than the SOTA model.

muscle activation (EMG), and heart rhythm (ECG). The PSG data is then analyzed by a trained sleep technician and a certified sleep doctor to produce a PSG report. The hypnogram plays an essential role in the PSG report, where it is used to derive many important metrics such as sleep efficiency and apnea index. Unfortunately, manually annotating this PSG is both costly and time consuming for the doctors. Recent research has proposed to alleviate these issues by automatically generating the hypnogram directly from the PSG using deep neural networks [317, 318]. However, the process of obtaining a PSG report is still *costly* and *invasive* to patients, reducing their participation, which ultimately leads to undiagnosed sleep disorders [319].

One promising direction to reduce undiagnosed sleep disorders is to enable sleep monitoring at the home using commercial wearables (e.g., Fitbit, Apple Watch, Emotiv) [320]. However, despite significant research advances, a recent study shows that wearables using a single sensor (e.g., single lead EEG) often have lower performance for sleep staging, indicating a large room for improvement [321].

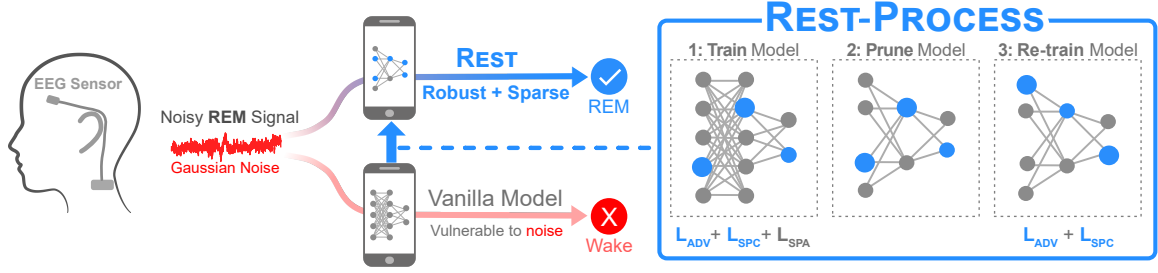


Figure 8.2: REST Overview: (from left) When a noisy EEG signal belonging to the REM (rapid eye movement) sleep stage enters a traditional neural network which is vulnerable to noise, it gets wrongly classified as a Wake sleep stage. On the other hand, the same signal is correctly classified as the REM sleep stage by the REST model which is both robust and sparse. (From right) REST is a three step process involving (1) training the model with adversarial training, spectral regularization and sparsity regularization (2) pruning the model and (3) re-training the compact model.

8.1.1 Contributions

Our contributions are two-fold—(i) we identify emerging research challenges for the task of sleep monitoring in the wild; and (ii) we propose REST, a novel framework that addresses these issues.

I. New Research Challenges for Sleep Monitoring.

- **C1. Robustness to Noise.** We observe that state-of-the-art deep neural networks (DNN) are highly susceptible to environmental noise (Fig. 8.1, top). In the case of wearables, noise is a serious consideration since bioelectrical signal sensors (e.g., electroencephalogram “EEG”, electrocardiogram “ECG”) are commonly susceptible to *Gaussian* and *shot* noise, which can be introduced by electrical interferences (e.g., power-line) and user motions (e.g., muscle contraction, respiration) [322, 323, 324, 325]. This poses a need for noise-tolerant models. We show that adversarial training and spectral regularization can impart significant noise robustness to sleep staging DNNs (see top of Fig 8.1).
- **C2. Energy and Computational Efficiency.** Mobile deep learning systems have traditionally offloaded compute intensive inference to cloud servers, requiring transfer of sensitive data and assumption of available Internet. However, this data uploading process is difficult for many healthcare scenarios because of—(1) **privacy**: individuals are often reluctant to share health information as they consider it highly sensitive; and (2) **accessibility**: real-time home monitoring is most needed in resource-poor environments where high-speed Internet may not be reliably available. Directly deploying a neural network to a mobile phone bypasses these issues. However, due to the constrained computation and energy budget of mobile devices, these models need to be fast in speed and parsimonious with their energy consumption.

II. Noise-robust and Efficient Sleep Monitoring. Having identified these two new research challenges, we propose **REST**, the first framework for developing noise-robust and efficient neural networks for home sleep monitoring (Fig. 8.2). Through REST, our major contributions include:

- **“Robust and Efficient Neural Networks for Sleep Monitoring”** By integrating a novel combination of three training objectives, REST endows a model with noise robustness through (1) *adversarial training* and (2) *spectral regularization*; and promotes energy and computational efficiency by enabling compression through (3) *sparsity regularization*.
- **Extensive evaluation** We benchmark the performance of REST against competitive baselines, on two real-world sleep staging EEG datasets—Sleep-EDF from Physionet and Sleep Heart Health Study (SHHS). We demonstrate that REST produces highly compact models that substantially outperform the original full-sized models in the presence of noise. REST models achieves a macro-F1 score of 0.67 vs. 0.39 for the state-of-the-art model in the presence of Gaussian noise, with $19\times$ parameter and $15\times$ MFLOPS reduction.
- **Real-world deployment.** We deploy a REST model onto a Pixel 2 smartphone through an Android application performing sleep staging. Our experiments reveal REST achieves $17\times$ energy reduction and $9\times$ faster inference on a smartphone, compared to uncompressed models.

8.2 Related Work

In this section we discuss related work from three areas—(1) the task of sleep stage prediction, (2) robustness of deep neural networks and (3) compression of deep learning models.

8.2.1 Sleep-Stage Prediction

Sleep staging is the task of annotating a polysomnography (PSG) report into a hypnogram, where 30 second sleep intervals are annotated into one of five sleep stages (W, N1, N2, N3, REM). Recently, significant effort has been devoted towards automating this annotation process using deep learning [316, 317, 326, 327, 328, 329], to name a few. While there exists a large body of research in this area—two works in particular look at both single channel [317] and multi-channel [326] deep learning architectures for sleep stage prediction on EEG. In [317], the authors develop a deep learning architecture (SLEEPNET) for sleep stage prediction that achieves expert-level accuracy on EEG data. In [326], the authors

develop a multi-modal deep learning architecture for sleep stage prediction that achieves state-of-the-art accuracy. As we demonstrate later in this chapter (Section 8.4.5), these sleep staging models are frequently susceptible to noise and suffer a large performance drop in its presence (see Figure 8.1). In addition, these DNNs are often overparameterized (Section 8.4.6), making deployment to mobile devices and wearables difficult. Through REST, we address these limitations and develop noise robust and efficient neural networks for edge computing.

8.2.2 *Noise & Adversarial Robustness*

Adversarial robustness seeks to ensure that the output of a neural network remains unchanged under a bounded perturbation of the input; or in other words, prevent an adversary from maliciously perturbing the data to fool a neural network. Adversarial deep learning was popularized by [330], where they showed it was possible to alter the class prediction of deep neural network models by carefully crafting an adversarially perturbed input. Since then, research suggests a strong link between adversarial robustness and noise robustness [331, 332, 286]. In particular, [331] found that by performing adversarial training on a deep neural network, it becomes robust to many forms of noise (e.g., Gaussian, blur, shot, etc.). In contrast, they found that training a model on Gaussian augmented data led to models that were less robust to adversarial perturbations. We build upon this finding of adversarial robustness as a proxy for noise robustness and improve upon it through the use of spectral regularization; while simultaneously compressing the model to a fraction of its original size for mobile devices.

8.2.3 *Model Compression*

Model compression aims to learn a reduced representation of the weights that parameterize a neural network; shrinking the computational requirements for memory, floating point operations (FLOPS), inference time and energy. Broadly, prior art can be classified into four directions—pruning [333], quantization [334], low rank approximation [335] and knowledge distillation [336]. For REST, we focus on structured (channel) pruning thanks to its performance benefits (speedup, FLOP reduction) and ease of deployment with regular hardware. In structured channel pruning, the idea is to assign a measure of importance to each filter of a convolutional neural network (CNN) and achieve desired sparsity by pruning the least important ones. Prior work demonstrates several ways to estimate filter importance—magnitude of weights [337], structured sparsity regularization [338], regularization on activation scaling factors [339], filter similarity [340] and discriminative power

of filters [341]. Recently there has been an attempt to bridge the area of model compression with adversarial robustness through connection pruning [342] and quantization [343]. Different from previous work, REST aims to compress a model by pruning whole filters while imparting noise tolerance through adversarial training and spectral regularization. REST can be further compressed through quantization [343].

8.3 REST: Noise-Robust & Efficient Models

REST is a new method that simultaneously compresses a neural network while developing both noise and adversarial robustness.

8.3.1 Overview

Our main idea is to enable REST to endow models with these properties by integrating three careful modifications of the traditional training loss function. (1) The *adversarial training* term, which builds noise robustness by training on adversarial examples (Section 8.3.2); (2) the *spectral regularization* term, which adds to the noise robustness by constraining the Lipschitz constant of the neural network (Section 8.3.3); and (3) the sparsity regularization term that helps to identify important neurons and enables compression (Section 8.3.4). Throughout the chapter, we follow standard notation and use capital bold letters for matrices (e.g., \mathbf{A}), lower-case bold letters for vectors (e.g., \mathbf{a}).

8.3.2 Adversarial Training

The goal of adversarial training is to generate noise robustness by exposing the neural network to adversarially perturbed inputs during the training process. Given a neural network $f(\mathbf{X}; \mathbf{W})$ with input \mathbf{X} , weights \mathbf{W} and corresponding loss function $L(f(\mathbf{X}; \mathbf{W}), y)$, adversarial training aims at solving the following min-max problem:

$$\min_{\mathbf{W}} \left[\mathbb{E}_{\mathbf{X}, y \sim D} \left(\max_{\delta \in S} L(f(\mathbf{X} + \delta; \mathbf{W}), y) \right) \right] \quad (8.1)$$

Here D is the unperturbed dataset consisting of the clean EEG signals $\mathbf{X} \in \mathbb{R}^{K_{in} \times K_L}$ (K_{in} is the number of channels and K_L is the length of the signal) along with their corresponding label y . The inner maximization problem in (8.1) embodies the goal of the adversary—that is, produce adversarially perturbed inputs (i.e., $\mathbf{X} + \delta$) that maximize the loss function L . On the other hand, the outer minimization term aims to build robustness by countering the adversary through minimizing the expected loss on perturbed inputs.

Maximizing the inner loss term in (8.1) is equivalent to finding the adversarial signal $\mathbf{X}_p = \mathbf{X} + \delta$ that maximally alters the loss function L within some bounded perturbation $\delta \in S$. Here S is the set of allowable perturbations. Several choices exist for such an adversary. For REST, we use the iterative Projected Gradient Descent (PGD) adversary since it's one of the strongest first order attacks [294]. Its operation is described below in Equation 8.2.

$$\mathbf{X}_p^{(t+1)} = \mathbf{X}_p^{(t)} + \Pi_\tau \left[\epsilon \cdot \text{sign} \left\{ \nabla_{\mathbf{X}_p^{(t)}} L(f(\mathbf{X}_p^{(t)}; \mathbf{W}), y) \right\} \right] \quad (8.2)$$

Here $\mathbf{X}_p^{(0)} = \mathbf{X}$ and at every step t , the previous perturbed input $\mathbf{X}_p^{(t-1)}$ is modified with the sign of the gradient of the loss, multiplied by ϵ (controls attack strength). Π_τ is a function that clips the input at the positions where it exceeds the predefined L_∞ bound τ . Finally, after n_{iter} iterations we have the REST adversarial training term L_{adv} in Equation 8.3.

$$L_{adv} = L(f(\mathbf{X}_p^{(n_{iter})}; \mathbf{W}), y) \quad (8.3)$$

8.3.3 Spectral Regularizer

The second term in the objective function is the spectral regularization term, which aims to constrain the change in output of a neural network for some change in input. The intuition is to suppress the amplification of noise as it passes through the successive layers of a neural network. In this section we show that an effective way to achieve this is via constraining the Lipschitz constant of each layer's weights.

For a real valued function $f : \mathbb{R} \rightarrow \mathbb{R}$ the Lipschitz constant is a positive real value C such that $|f(x_1) - f(x_2)| \leq C|x_1 - x_2|$. If $C > 1$ then the change in input is magnified through the function f . For a neural net, this can lead to input noise amplification. On the other hand, if $C < 1$ then the noise amplification effect is diminished. This can have the unintended consequence of reducing the discriminative capability of a neural net. Therefore our goal is to set the Lipschitz constant $C = 1$. The Lipschitz constant for the l^{th} fully connected layer parameterized by the weight matrix $\mathbf{W}^{(l)} \in \mathbb{R}^{K_{in} \times K_{out}}$ is equivalent to its spectral norm $\rho(\mathbf{W}^{(l)})$ [344]. Here the spectral norm of a matrix \mathbf{W} is the square root of the largest singular value of $\mathbf{W}^T \mathbf{W}$. The spectral norm of a 1-D convolutional layer parameterized by the tensor $\mathbf{W}^{(l)} \in \mathbb{R}^{K_{out} \times K_{in} \times K_l}$ can be realized by reshaping it to a matrix $\mathbf{W}^{(l)} = \mathbb{R}^{K_{out} \times (K_{in} K_l)}$ and then computing the largest singular value.

A neural network of N layers can be viewed as a function $f(\cdot)$ composed of N sub-functions $f(x) = f_1(\cdot) \circ f_2(\cdot) \circ \dots \circ f_N(x)$. A loose upper bound for the Lipschitz constant

Algorithm 4: Noise Robust & Efficient Neural Network Training (REST)

Input: Model weights \mathbf{W} , EEG signal \mathbf{X} and label y from dataset \mathbf{D} , spectral regularization λ_o , sparsity regularization λ_g , learning rate α , perturbation strength ϵ , maximum PGD iterations n_{iter} and model sparsity s

Output: Noise robust, compressed neural network

1 (1) **Train the full model with REST loss L_R :**

2 **for** $epoch = 1$ to N **do**

3 **for** minibatch $B \subset D$ **do**

4 **for** $X \in B$ **do**

5 $\mathbf{X}_p^{(1)} = \mathbf{X}$

6 **for** $k=1$ to n_{iter} **do**

7 $\mathbf{X}_p^{(k+1)} = \mathbf{X}_p^{(k)} + \Pi_\tau(\epsilon \cdot \text{sign}(\nabla_{\mathbf{X}_p^{(k)}} L(f(\mathbf{X}_p^{(k)}; \mathbf{W}), y)))$

8 $\mathbf{W}_{grad} \leftarrow \mathbb{E}_{\mathbf{X}, y \sim D} |\nabla_{\mathbf{W}} L_R(\mathbf{X}_p, y; \mathbf{W})|$

9 where $L_R =$

$$\underbrace{L(f(\mathbf{X}_p; \mathbf{W}), y)}_{\text{adversarial training}} + \underbrace{\lambda_o \sum_{\text{layer } l=1}^N \|(\mathbf{W}^{(l)})^T \mathbf{W}^{(l)} - \mathbf{I}\|_2}_{\text{spectral regularization}} + \underbrace{\lambda_g \sum_{\text{layer } l=1}^N \|\gamma^{(l)}\|_1}_{\text{sparsity regularization}}$$

10 $\mathbf{W} \leftarrow \mathbf{W} - \alpha \cdot \mathbf{W}_{grad}$

11 (2) **Prune the trained model:**

12 Globally prune filters from \mathbf{W} having smallest γ values until $\frac{n_f(\mathbf{W}')}{n_f(\mathbf{W})} \leq s$. Constrain layerwise sparsity so $\frac{n_f(\mathbf{W}'^{(l)})}{n_f(\mathbf{W}^{(l)})} \geq 0.1$.

13 (3) **Re-train the pruned model:**

14 Retrain compressed network $f(\mathbf{X}; \mathbf{W}')$ using *adversarial training* and *spectral regularization* (no sparsity regularization).

of f is the product of Lipschitz constants of individual layers or $\rho(f) \leq \prod_{i=1}^N \rho(f_i)$ [344]. The overall Lipschitz constant can grow exponentially if the spectral norm of each layer is greater than 1. On the contrary, it could go to 0 if spectral norm of each layer is between 0 and 1. Thus the ideal case arises when the spectral norm for each layer equals 1. This can be achieved in several ways [345, 344, 346], however, one effective way is to encourage orthonormality in the columns of the weight matrix \mathbf{W} through the minimization of $\|\mathbf{W}^T \mathbf{W} - \mathbf{I}\|$ where \mathbf{I} is the identity matrix. This additional loss term helps regulate the singular values and bring them close to 1. Thus we incorporate the following spectral regularization term into our loss objective, where λ_o is a hyperparameter controlling the

strength of the spectral regularization.

$$L_{Spectral} = \lambda_o \sum_{i=1}^N \|(\mathbf{W}^{(i)})^T \mathbf{W}^{(i)} - \mathbf{I}\|_2 \quad (8.4)$$

8.3.4 Sparsity Regularizer & REST Loss Function

The third term of the REST objective function consists of the sparsity regularizer. With this term, we aim to learn the important filters in the neural network. Once these are determined, the original neural network can be pruned to the desired level of sparsity.

The incoming weights for filter i in the l^{th} fully connected (or 1-D convolutional) layer can be specified as $\mathbf{W}_{i,:}^{(l)} \in \mathbb{R}^{K_{in}}$ (or $\mathbf{W}_{i,:,:}^{(l)} \in \mathbb{R}^{K_{in} \times K_L}$). We introduce a per filter multiplicand $\gamma_i^{(l)}$ that scales the output activation of the i^{th} neuron in layer l . By controlling the value of this multiplicand, we realize the importance of the neuron. In particular, zeroing it amounts to dropping the entire filter. Note that the L_0 norm on the multiplicand vector $\|\gamma^{(l)}\|_0$, where $\gamma^{(l)} \in \mathbb{R}^{K_{out}}$, can naturally satisfy the sparsity objective since it counts the number of non zero entries in a vector. However since the L_0 norm is a nondifferentiable function, we use the L_1 norm as a surrogate [347, 338, 339] which is amenable to backpropagation through its subgradient.

To realize the per filter multiplicand $\gamma_i^{(l)}$, we leverage the per filter multiplier within the batch normalization layer [339]. In most modern networks, a batchnorm layer immediately follows the convolutional/linear layers and implements the following operation.

$$\mathbf{B}_i^{(l)} = \left(\frac{\mathbf{A}_i^{(l)} - \boldsymbol{\mu}_i^{(l)}}{\boldsymbol{\sigma}_i^{(l)}} \right) \gamma_i^{(l)} + \boldsymbol{\beta}_i^{(l)} \quad (8.5)$$

Here $\mathbf{A}_i^{(l)}$ denotes output activation of filter i in layer l while $\mathbf{B}_i^{(l)}$ denotes its transformation through batchnorm layer l ; $\boldsymbol{\mu}^{(l)} \in \mathbb{R}^{K_{out}}$, $\boldsymbol{\sigma}^{(l)} \in \mathbb{R}^{K_{out}}$ denote the mini-batch mean and standard deviation for layer l 's activations; and $\gamma^{(l)} \in \mathbb{R}^{K_{out}}$ and $\boldsymbol{\beta}^{(l)} \in \mathbb{R}^{K_{out}}$ are learnable parameters. Our sparsity regularization is defined on $\gamma^{(l)}$ as below, where λ_g is a hyperparameter controlling the strength of sparsity regularization.

$$L_{Sparsity} = \lambda_g \sum_{i=1}^N \|\gamma^{(l)}\|_1 \quad (8.6)$$

The sparsity regularization term (8.6) promotes learning a subset of important filters while training the model. Compression then amounts to globally pruning filters with the smallest value of multiplicands in (8.5) to achieve the desired model compression. Pruning

typically causes a large drop in accuracy. Once the pruned model is identified, we fine-tune it via retraining.

Now that we have discussed each component of REST, we present the full loss function in (8.7) and the training process in Algorithm 4. A pictorial overview of the process can be seen in Figure 8.2.

$$\begin{aligned}
L_R = & \underbrace{L(f(\mathbf{X}_p; \mathbf{W}), y)}_{\text{adversarial training}} + \underbrace{\lambda_o \sum_{i=1}^N \|(\mathbf{W}^{(i)})^T \mathbf{W}^{(i)} - \mathbf{I}\|_2}_{\text{spectral regularization}} \\
& + \underbrace{\lambda_g \sum_{i=1}^N \|\gamma^{(i)}\|_1}_{\text{sparsity regularization}}
\end{aligned} \tag{8.7}$$

8.4 Experiments

We compare the efficacy of REST neural networks to four baseline models (Section 8.4.2) on two publicly available EEG datasets—Sleep-EDF from Physionet [348] and Sleep Heart Health Study (SHHS) [349]. Our evaluation focuses on two broad directions—**noise robustness** and **model efficiency**. Noise robustness compares the efficacy of each model when EEG data is corrupted with three types of noise: *adversarial*, *Gaussian* and *shot*. Model efficiency compares both static (e.g., model size, floating point operations) and dynamic measurements (e.g., inference time, energy consumption). For dynamic measurements which depend on device hardware, we deploy each model to a Pixel 2 smartphone.

8.4.1 Datasets

Our evaluation uses two real-world sleep staging EEG datasets.

- **Sleep-EDF:** This dataset consists of data from two studies—age effect in healthy subjects (SC) and Temazepam effects on sleep (ST). Following [318], we use whole-night polysomnographic sleep recordings on 40 healthy subjects (one night per patient) from

Table 8.1: Dataset summary outlining the number of 30 second EEG recordings belonging to each sleep stage class.

Dataset	W	N1	N2	N3(N4)	REM	Total
Sleep-EDF	8,168	2,804	17,799	5,703	7,717	42,191
SHHS	28,854	3,377	41,246	13,409	13,179	100,065

SC. It is important to note that the SC study is conducted in the subject’s homes, not a sleep center and hence this dataset is inherently noisy. However, the sensing environment is still relatively controlled since sleep doctors visited the patient’s home to setup the wearable EEG sensors. After obtaining the data, the recordings are manually classified into one of eight classes (W, N1, N2, N3, N4, REM, MOVEMENT, UNKNOWN); we follow the steps in [318] and merge stages N3 and N4 into a single N3 stage and exclude MOVEMENT and UNKNOWN stages to match the five stages of sleep according to the American Academy of Sleep Medicine (AASM) [350]. Each single channel EEG recording of 30 seconds corresponds to a vector of dimension 1×3000 . Similar to [316], while scoring at time i , we include EEG recordings from times $i - 3, i - 2, i - 1, i$. Thus we expand the EEG vector by concatenating the previous three time steps to create a vector of size 1×12000 . After pre-processing the data, our dataset consists of 42,191 EEG recordings, each described by a 12,000 length vector and assigned a sleep stage label from Wake, N1, N2, N3 and REM using the Fpz-Cz EEG sensor (see Table 8.1 for sleep stage breakdown). Following standard practice [318], we divide the dataset on a *per-patient, whole-night* basis, using 80% for training, 10% for validation, and 10% for testing. That is, a single patient is recorded for one night and can only be in one of the three sets (training, validation, testing). The final number of EEG recordings in their respective splits are 34,820, 5,345 and 3,908. While the number of recordings appear to differ from the 80-10-10 ratio, this is because the data is split over the total number of *patients*, where each patient is monitored for a time period of variable length (9 hours \pm few minutes.)

- **Sleep Heart Health Study (SHHS):** The Sleep Heart Health Study consists of two rounds of polysomnographic recordings (SHHS-1 and SHHS-2) sampled at 125 Hz in a sleep center environment. Following [316], we use only the first round (SHHS-1) containing 5,793 polysomnographic records over two channels (C4-A1 and C3-A2). Recordings are manually classified into one of six classes (W, N1, N2, N3, N4 and REM). As suggested in [350], we merge N3 and N4 stages into a single N3 stage (see Table 8.1 for sleep stage breakdown). We use 100 distinct patients randomly sampled from the original dataset (one night per patient). Similar to [316], we look at three previous time steps in order to score the EEG recording at the current time step. This amounts to concatenating the current EEG recording of size 1×3750 (equal to 125 Hz \times 30 Hz) to generate an EEG recording of size 1×15000 . After this pre-processing, our dataset consists of 100,065 EEG recordings, each described by a 15,000 length vector and assigned a sleep stage label from the same 5 classes using the Fpz-Cz EEG sensor. We use the same 80-10-10 data split as in Sleep-EDF, resulting in 79,940 EEG recordings for training, 9,999

for validation, and 10,126 for testing.

8.4.2 Model Architecture and Configurations

We use the sleep staging CNN architecture proposed by [316], since it achieves state-of-the-art accuracy for sleep stage classification using single channel EEG. We implement all models in PyTorch 0.4. For training and evaluation, we use a server equipped with an Intel Xeon E5-2690 CPU, 250GB RAM and 8 Nvidia Titan Xp GPUs. Mobile device measurements use a Pixel 2 smartphone with an Android application running Tensorflow Lite¹. With [316] as the architecture for all baselines below, we compare the following 6 configurations:

1. **Sors** [316]: Baseline neural network model trained on unperturbed data. This model contains 12 1-D convolutional layers followed by 2 fully connected layers and achieves state-of-the-art performance on sleep staging using single channel EEG.
2. **Liu** [339]: We train on unperturbed data and compress the Sors model using sparsity regularization as proposed in [339].
3. **Blanco** [351]: We use same setup from Liu above. During test time, the noisy test input is filtered using a bandpass filter with cutoff 0.5Hz-40Hz This technique is commonly used for removing noise in EEG analysis [351].
4. **Ford** [331]: We train and compress the Sors model with sparsity regularization on input data perturbed by Gaussian noise. Gaussian training parameter $c_g = 0.2$ controls the perturbation strength during training; identified through a line search in Section 8.4.4.
5. **REST (A)**: Our compressed Sors model obtained through adversarial training and sparsity regularization. We use the hyperparameters: $\epsilon = 10$, $n_{iter} = 5/10$ (SHHS/Sleep-EDF), where ϵ is a *key* variable controlling the strength of adversarial perturbation during training. The optimal ϵ value is determined through a line search described in Section 8.4.4.
6. **REST (A+S)**: Our compressed Sors model obtained through adversarial training, spectral and sparsity regularization. We set the spectral regularization parameter $\lambda_o = 3 \times 10^{-3}$ and sparsity regularization parameter $\lambda_g = 10^{-5}$ based on a grid search in Section 8.4.4.

¹TensorFlow Lite: <https://www.tensorflow.org/lite>

All models are trained for 30 epochs using SGD. The initial learning rate is set to 0.1 and multiplied by 0.1 at epochs 10 and 20; the weight decay is set to 0.0002. All compressed models use the same compression method, consisting of weight pruning followed by model re-training. The sparsity regularization parameter $\lambda_g = 10^{-5}$ is identified through a grid search with λ_o (after determining ϵ through a line search). Detailed analysis of the hyperparameter selection for ϵ , λ_o and λ_g can be found in Section 8.4.4. Finally, we set a high sparsity level $s = 0.8$ (80% neurons from the original networks were pruned) after observation that the models are overparametrized for the task of sleep stage classification.

8.4.3 Evaluation Metrics

Noise robustness metrics To study the noise robustness of each model configuration, we evaluate macro-F1 score in the presence of three types of noise: adversarial, Gaussian and shot. We select macro-F1 since it is a standard metric for evaluating classification performance in imbalanced datasets. Adversarial noise is defined at three strength levels through $\epsilon = 2/6/12$ in Equation 8.2; Gaussian noise at three levels through $c_g = 0.1/0.2/0.3$ in Equation 8.8; and shot noise at three levels through $c_s = 5000/2500/1000$ in Equation 8.9. These parameter values are chosen based on prior work [294, 332] and empirical observation. For evaluating robustness to adversarial noise, we assume the white box setting where the attacker has access to model weights. The formulation for Gaussian and shot noise is in Equation 8.8 and 8.9, respectively.

$$\mathbf{X}_{gauss} = \mathbf{X} + N(0, c_g \cdot \sigma_{train}) \quad (8.8)$$

In Equation 8.8, σ_{train} is the standard deviation of the training data and N is the normal distribution. The noise strength—low, medium and high—corresponds to $c_g = 0.1/0.2/0.3$.

$$\begin{aligned} \mathbf{X}_{norm} &= \frac{\mathbf{X} - x_{min}}{x_{max} - x_{min}} \\ \mathbf{X}' &= clip_{0,1} \left(\frac{Poisson(\mathbf{X}_{norm} \cdot c_s)}{c_s} \right) \\ \mathbf{X}_{shot} &= \mathbf{X}' \cdot (x_{max} - x_{min}) + x_{min} \end{aligned} \quad (8.9)$$

In Equation 8.9, x_{min}, x_{max} denote the minimum and maximum values in the training data; and $clip_{0,1}$ is a function that projects the input to the range $[0,1]$.

Model efficiency metrics To evaluate the efficiency of each model configuration, we use the following measures:

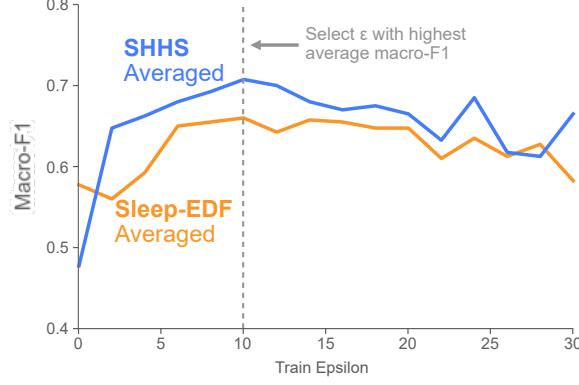


Figure 8.3: Line search results for ϵ on Sleep-EDF and SHHS datasets. We select $\epsilon=10$, since it provides the best average macro-F1 score on both datasets.

- **Parameter Reduction:** Memory consumed (in KB) for storing the weights of a model.
- **Floating point operations (FLOPS):** Number of multiply and add operations performed by the model in one forward pass. Measurement units are Mega (10^6).
- **Inference Time:** Average time taken (in seconds) to score one night of EEG data. We assume a night consists of 9 hours and amounts to 1,080 EEG recordings (each of 30 seconds). This is measured on a Pixel 2 smartphone.
- **Energy Consumption:** Average energy consumed by a model (in Joules) to score one night of EEG data on a Pixel 2 smartphone. To measure consumed energy, we implement an infinite inference loop over EEG recordings until the battery level drops from 100% down to 85%. For each unit percent drop (i.e., 15 levels), we log the number of iterations N_i performed by the model. Given that a standard Pixel 2 battery can deliver 2700 mAh at 3.85 Volts, we use the following conversion to estimate energy consumed E (in Joules) for a unit percent drop in battery level $E = \frac{2700}{1000} \times 3600 \times 3.85$. The total energy for inferencing over an entire night of EEG recordings is then calculated as $\frac{E}{N_i} \times 1080$ where N_i is the number of inferences made in the unit battery drop interval. We average this for every unit battery percentage drop from 100% to 85% (i.e., 15 intervals) to calculate the average energy consumption

8.4.4 Hyperparameter Selection

Optimal hyper-parameter selection is crucial for obtaining good performance with both baseline and REST models. We systematically conduct a series of line and grid searches to determine ideal values of ϵ , c_g , λ_o and λ_g using the validation sets.

Selecting ϵ This parameter controls the perturbation strength of adversarial training in

Table 8.2: Line search results for identifying optimal c_g on Sleep-EDF and SHHS datasets. Macro-F1 is abbreviated F1 in table; average macro-F1 is the mean of all macro-F1 scores. We select $c_g=0.2$ for both datasets as it represents a good trade-off between benign and Gaussian macro-F1.

	c_g	Benign F1	Gaussian F1			Average F1
			Low	Med	High	
EDF	0.1	0.75	0.76	0.7	0.5	0.68
	0.2	0.7	0.72	0.75	0.64	0.70
	0.3	0.67	0.68	0.71	0.75	0.7025
SHHS	0.1	0.69	0.74	0.45	0.21	0.52
	0.2	0.68	0.69	0.68	0.43	0.62
	0.3	0.55	0.57	0.65	0.74	0.63

Equation 8.2. Correctly setting this parameter is critical since a small ϵ value will have no effect on noise robustness, while too high a value will lead to poor benign accuracy. We follow standard procedure and determine the optimal ϵ on a per-dataset basis [294], conducting a line search across $\epsilon \in [0,30]$ in steps of 2. For each value of ϵ we measure benign and adversarial validation macro-F1 score, where adversarial macro-F1 is an average of three strength levels: low ($\epsilon=2$), medium ($\epsilon=6$) and high ($\epsilon=12$). We then select the ϵ with highest macro-F1 score averaged across the benign and adversarial macro-F1. Line search results are shown in Figure 8.3; we select $\epsilon = 10$ for both dataset since it's the value with highest average macro-F1.

Table 8.3: Grid search results for λ_o and λ_g on Sleep-EDF dataset. Macro-F1 is abbreviated as F1 in table; average macro-F1 is the mean of all macro-F1 scores. We select λ_o and λ_g with highest average macro-F1 score.

λ_o	λ_g	Benign F1	Adversarial F1			Avg. F1
			Low	Med	High	
0.001	1E-04	0.73	0.66	0.65	0.61	0.66
0.003	1E-04	0.72	0.64	0.63	0.59	0.65
0.005	1E-04	0.72	0.65	0.64	0.62	0.66
0.001	1E-05	0.73	0.66	0.65	0.62	0.67
0.003	1E-05	0.73	0.67	0.66	0.62	0.67
0.005	1E-05	0.73	0.64	0.64	0.62	0.66

Selecting c_g This parameter controls the noise perturbation strength of Gaussian training in Equation 8.8. Similar to ϵ , we determine c_g on a per-dataset basis, conducting a line search across c_g values: 0.1 (low), 0.2 (medium) and 0.3 (high). Based on results from

Table 8.4: Meta Analysis: Comparison of macro-F1 scores achieved by each model. The models are evaluated on Sleep-EDF and SHHS datasets with three types and strengths of noise corruption. We bold the compressed model with the best performance (averaged over 3 runs) and report the standard deviation of each model next to the macro-F1 score. REST performs better in *all* noise test measurements.

Data	Method	Compress	No noise	Adversarial			Gaussian			Shot		
				Low	Med	High	Low	Med	High	Low	Med	High
Sleep-EDF	Sors [316]	\times	0.67 ± 0.02	0.57 ± 0.02	0.51 ± 0.04	0.19 ± 0.06	0.66 ± 0.03	0.60 ± 0.03	0.39 ± 0.08	0.58 ± 0.04	0.42 ± 0.08	0.11 ± 0.03
	Liu [339]	\checkmark	0.69 ± 0.02	0.52 ± 0.07	0.41 ± 0.07	0.09 ± 0.02	0.67 ± 0.02	0.53 ± 0.02	0.28 ± 0.04	0.52 ± 0.03	0.31 ± 0.04	0.06 ± 0.01
	Blanco [351]	\checkmark	0.68 ± 0.01	0.51 ± 0.06	0.40 ± 0.06	0.09 ± 0.02	0.65 ± 0.02	0.54 ± 0.04	0.31 ± 0.10	0.53 ± 0.04	0.34 ± 0.09	0.08 ± 0.02
	Ford [331]	\checkmark	0.64 ± 0.01	0.59 ± 0.01	0.60 ± 0.02	0.31 ± 0.08	0.65 ± 0.01	0.67 ± 0.02	0.57 ± 0.03	0.67 ± 0.02	0.60 ± 0.02	0.10 ± 0.01
	REST (A)	\checkmark	0.66 ± 0.02	0.64 ± 0.02	0.64 ± 0.02	0.61 ± 0.02	0.66 ± 0.02	0.67 ± 0.01	0.66 ± 0.01	0.67 ± 0.01	0.66 ± 0.01	0.42 ± 0.06
	REST (A+S)	\checkmark	0.69 ± 0.01	0.67 ± 0.02	0.66 ± 0.01	0.61 ± 0.03	0.69 ± 0.01	0.68 ± 0.01	0.67 ± 0.02	0.68 ± 0.01	0.67 ± 0.02	0.42 ± 0.08
SHHS	Sors [316]	\times	0.78 ± 0.01	0.62 ± 0.03	0.46 ± 0.03	0.33 ± 0.00	0.64 ± 0.03	0.43 ± 0.02	0.35 ± 0.04	0.69 ± 0.02	0.59 ± 0.03	0.45 ± 0.01
	Liu [339]	\checkmark	0.77 ± 0.01	0.61 ± 0.02	0.49 ± 0.04	0.34 ± 0.03	0.66 ± 0.05	0.45 ± 0.05	0.34 ± 0.04	0.70 ± 0.04	0.62 ± 0.04	0.47 ± 0.05
	Blanco [351]	\checkmark	0.77 ± 0.01	0.60 ± 0.03	0.47 ± 0.04	0.33 ± 0.02	0.64 ± 0.07	0.43 ± 0.05	0.34 ± 0.04	0.67 ± 0.06	0.59 ± 0.05	0.46 ± 0.04
	Ford [331]	\checkmark	0.62 ± 0.02	0.59 ± 0.01	0.62 ± 0.00	0.59 ± 0.05	0.66 ± 0.00	0.75 ± 0.04	0.47 ± 0.10	0.65 ± 0.00	0.68 ± 0.01	0.74 ± 0.04
	REST (A)	\checkmark	0.70 ± 0.01	0.68 ± 0.00	0.70 ± 0.01	0.67 ± 0.01	0.72 ± 0.01	0.76 ± 0.01	0.58 ± 0.03	0.72 ± 0.01	0.74 ± 0.01	0.76 ± 0.01
	REST (A+S)	\checkmark	0.72 ± 0.01	0.69 ± 0.01	0.70 ± 0.01	0.69 ± 0.02	0.74 ± 0.01	0.77 ± 0.01	0.62 ± 0.03	0.73 ± 0.01	0.75 ± 0.01	0.78 ± 0.00

Table 8.2, we select $c_g=0.2$ for both datasets since it provides the best average macro-F1 score while minimizing the drop in benign accuracy.

Selecting λ_o and λ_g These parameters determine the strength of spectral and sparsity regularization in Equation 8.7. We determine the best value for λ_o and λ_g through a grid search across the following parameter values $\lambda_o = [0.001, 0.003, 0.005]$ and $\lambda_g = [1E - 04, 1E - 05]$. Based on results from Table 8.3, we select $\lambda_o = 0.003$ and $\lambda_g = 1E - 05$. Since these are model dependent parameters, we calculate them once on the Sleep-EDF dataset and re-use them for SHHS.

8.4.5 Noise Robustness

To evaluate noise robustness, we ask the following questions—(1) what is the impact of REST on model accuracy with and without noise in the data? and (2) how does REST training compare to baseline methods of benign training, Gaussian training and noise filtering? In answering these questions, we analyze noise robustness of models at three scales: (i) meta-level macro-F1 scores; (ii) meso-level confusion matrix heatmaps; and (iii) granular-level single-patient hypnograms.

I. Meta analysis: Macro-F1 Scores In Table 8.4, we present a high-level overview of model performance through macro-F1 scores on three types and strength levels of noise corruption. The Macro-F1 scores and standard deviation are reported by averaging over three runs for each model and noise level. We identify multiple key insights as described below:

1. **REST Outperforms Across All Types of Noise** As demonstrated by the higher macro-F1 scores, REST outperforms all baseline methods in the presence of noise. In addition, REST has a low standard deviation, indicating model performance is not dependent on weight initialization.
2. **Spectral Regularization Improves Performance** REST ($A + S$) consistently improves upon REST (A), indicating the usefulness of spectral regularization towards enhancing noise robustness by constraining the Lipschitz constant.
3. **SHHS Performance Better Than Sleep-EDF** Performance is generally better on the SHHS dataset compared to Sleep-EDF. One possible explanation is due to the SHHS dataset being less noisy in comparison to the Sleep-EDF dataset. This stems from the fact that the SHHS study was performed in the hospital setting while Sleep-EDF was undertaken in the home setting.
4. **Benign & Adversarial Accuracy Trade-off** Contrary to the traditional trade-off between benign and adversarial accuracy, REST performance matches Liu in the no noise setting on sleep-EDF. This is likely attributable to the noise in the Sleep-EDF dataset, which was collected in the home setting. On the SHHS dataset, the Liu model outperforms REST in the no noise setting, where data is captured in the less noise prone hospital setting. Due to this, REST models are best positioned for use in noisy environments (e.g., at home); while traditional models are more effective in controlled environments (e.g., sleep labs).

II. Meso Analysis: Per-class Performance We visualize and identify class-wise trends using confusion matrix heatmaps (Fig. 8.4). Each confusion matrix describes a model’s performance for a given level of noise (or no noise). A model that is performing well should have a dark diagonal and light off-diagonal. We normalize the rows of each confusion matrix to accurately represent class predictions in an imbalanced dataset. When a matrix diagonal has a value of 1 (dark blue, or dark green) the model predicts every example correctly; the opposite occurs at 0 (white). Analyzing Figure 8.4, we identify the following key insights:

1. **REST Performs Well Across All Classes** REST accurately predicts each sleep stage (W, N1, N2, N3, REM) across multiple types of noise (Fig. 8.4, bottom 3 rows), as evidenced by the dark diagonal. In comparison, each baseline method has considerable performance degradation (light diagonal) in the presence of noise. This is particularly evident on the Sleep-EDF dataset (left half) where data is collected in the noisier home environment.

2. **N1 Class Difficult to Predict** When no noise is present (Fig. 8.4, top row), each method performs well as evidenced by the dark diagonal, except on the N1 sleep stage class. This performance drop is likely due to the limited number of N1 examples in the datasets (see Table 8.1).
3. **Increased Misclassification Towards “Wake” Class** On the Sleep-EDF dataset, shot and adversarial noise cause the baseline models to mispredict classes as Wake. One possible explanation is that the models misinterpret the additive noise as evidence for the wake class which has characteristically large fluctuations.

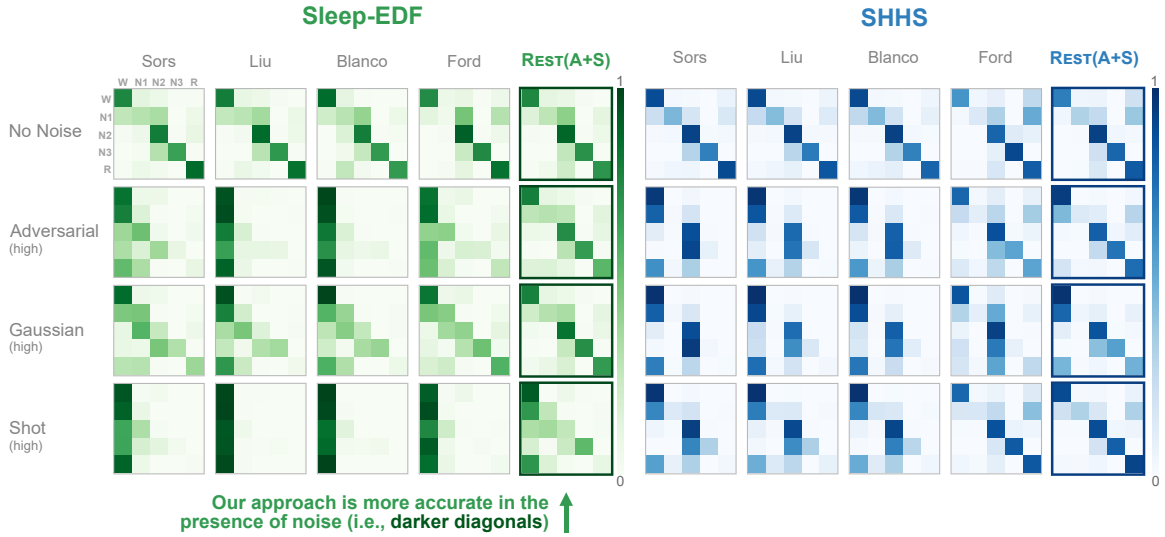


Figure 8.4: Meso Analysis: Class-wise comparison of model predictions. The models are evaluated over the SHHS test set perturbed with different noise types. In each confusion matrix, rows are ground-truth classes while columns are predicted classes. The intensity of a cell is obtained by normalizing the score with respect to the class membership. When a cell has a value of 1 (dark blue, or dark green) the model predicts every example correctly, the opposite occurs at 0 (white). A model that is performing well would have a dark diagonal and light off-diagonal. REST has the darkest cells along the diagonal on both datasets.

III. Granular Analysis: Single-patient Hypnograms We want to more deeply understand how our REST models counteract noise at the hypnogram level. Therefore, we select a test set patient from the SHHS dataset, and generate and visualize the patient’s overnight hypnograms using the Sors and REST models on three levels of Gaussian noise corruption (Figure 8.5). Each of these hypnograms is compared to a trained technicians hypnogram (expert scored in Fig. 8.5), representing the ground-truth. We inspect a few more test set patients using the above approach, and identify multiple key representative insights:

1. **Noisy Environments Require Robust Models** As data noise increases, Sors performance degrades. This begins at the low noise level, further accelerates in the medium

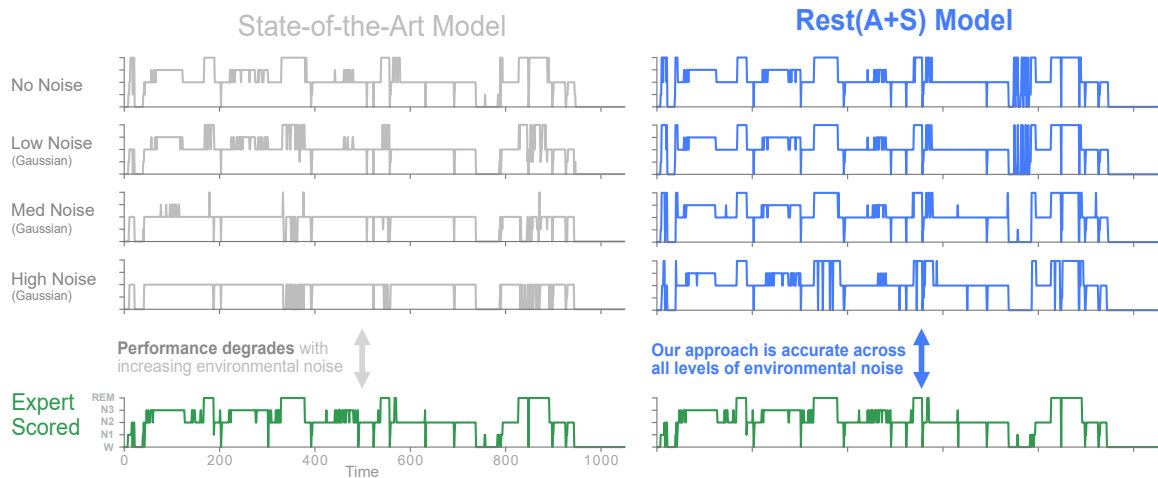


Figure 8.5: Granular Analysis: Comparison of the overnight hypnograms obtained for a patient in the SHHS test set. The hypnograms are generated using the Sors (left) and REST (right) models in the presence of increasing strengths of Gaussian noise. When no noise is present (top row), both models perform well, closely matching the ground truth (bottom row). However, with increasing noise, Sors performance rapidly degrades, while REST continues to generate accurate hypnograms.

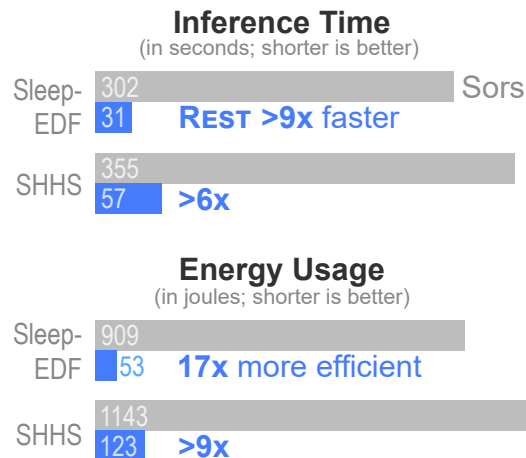


Figure 8.6: Time and energy consumption for scoring a single night of EEG recordings. REST(A+S) is significantly faster and more energy efficient than the state-of-the-art Sors model. Evaluations were done on a Pixel 2 smartphone.

level and reaches nearly zero at the high level. In contrast, REST effectively handles all levels of noise, generating an accurate hypnogram at even the highest level.

- 2. Low Noise Environments Give Good Performance** In the no noise setting (top row) both the Sors and REST models generate accurate hypnograms, closely matching the contours of expert scoring (bottom).

Table 8.5: Comparison on model size and the FLOPS required to score a single night of EEG recordings. REST models are significantly smaller and comparable in size/compute to baselines.

Data	Model	Size (KB)	MFlops
Sleep-EDF	Sors [316]	8,896	1451
	Liu [339]	440	127
	Blanco [351]	440	127
	Ford [331]	448	144
	REST (A)	464	98
	REST (A+S)	449	94
SHHS	Sors [316]	8,996	1815
	Liu [339]	464	211
	Blanco [351]	464	211
	Ford [331]	478	170
	REST (A)	476	160
	REST (A+S)	496	142

8.4.6 Model Efficiency

We measure model efficiency along two dimensions—(1) *static metrics*: amount of memory required to store weights in memory and FLOPS; and (2) *dynamic metrics*: inference time and energy consumption. For dynamic measurements that depend on device hardware, we deploy each model to a Pixel 2 smartphone.

Analyzing Static Metrics: Memory & Flops Table 8.5 describes the size (in KB) and computational requirements (in MFlops) of each model. We identify the following key insights:

1. **REST Models Require Fewest FLOPS** On both datasets, REST requires the least number of FLOPS.
2. **REST Models are Small** REST models are also smaller (or comparable) to baseline compressed models while achieving significantly better noise robustness.
3. **Model Efficiency and Noise Robustness** Combining the insights from Section 8.4.5 and the above, we observe that REST models have significantly better noise robustness while maintaining a competitive memory footprint. This suggests that robustness is more dependent on the the training process, rather than model capacity.

Analyzing Dynamic Metrics: Inference Time & Energy In Figure 8.6, we benchmark the inference time and energy consumption of a Sors and REST model deployed on a Pixel 2 smartphone using Tensorflow Lite. We identify the following insights:

1. **REST Models Run Faster** When deployed, REST runs $9\times$ and $6\times$ faster than the un-

compressed model on the two datasets.

2. **REST Models are Energy Efficient** REST models also consume $17\times$ and $9\times$ less energy than an uncompressed model on the Sleep-EDF and SHHS datasets, respectively.
3. **Enabling Sleep Staging for Edge Computing** The above benefits demonstrate that model compression effectively translates into faster inference and a reduction in energy consumption. These benefits are crucial for deploying on the edge.

8.5 Conclusion

We identified two key challenges in developing deep neural networks for sleep monitoring in the home environment—*robustness to noise* and *efficiency*. We proposed to solve these challenges through REST—a new method that simultaneously tackles both issues. For the sleep staging task over electroencephalogram (EEG), REST trains models that achieve up to $19\times$ parameter reduction and $15\times$ MFLOPS reduction with an increase of up to 0.36 in macro-F-1 score in the presence of noise. By deploying these models to a smartphone, we demonstrate that REST achieves up to $17\times$ energy reduction and $9\times$ faster inference.

Part V

Conclusions

CHAPTER 9

CONCLUSION AND FUTURE DIRECTIONS

This dissertation addresses the applied and theoretical challenges facing us in healthcare and cybersecurity—two of the most high impact domains—by developing new graph mining algorithms, deep learning models, large-scale databases, and open-source code. My research advances the frontiers of our technical understanding of data-driven disciplines in cybersecurity and healthcare, empowering people to make new discoveries and advances that positively impact millions of people across the world.

9.1 Research Contributions

New graph algorithms and deep learning models.

- We construct D²M, the first graph theoretic framework that systematically quantifies network vulnerability to lateral attack and identifies at-risk devices (Chapter 4).
- We develop REST, the first noise-robust and efficient deep learning model designed for at-home sleep monitoring by endowing models with noise robustness through (1) *adversarial training* and (2) *spectral regularization*; and promoting energy and computational efficiency by enabling compression through (3) *sparsity regularization* (Chapter 8).
- We contribute UNMASK, the first deep learning framework using semantic coherence to detect and defeat adversarial attacks by quantifying the similarity between the image’s extracted features with the expected features of its predicted class (Chapter 7).

Large-Scale Databases.

- We introduce MALNET-GRAPH, the largest public graph database ever constructed, representing a large-scale ontology of software function call graphs. MALNET-GRAPH contains over 1.2 million graphs, averaging over 17k nodes and 39k edges per graph, across a hierarchy of 47 types and 696 families. Compared to the popular REDDIT-12K database, MALNET-GRAPH offers 105× more graphs, 44× larger graphs on average, and 63× more classes (Chapter 5).
- We introduce MALNET-IMAGE, the largest publicly available cybersecurity image database, offering 24× more images and 70× more classes than existing databases. The scale and diversity of MALNET-IMAGE unlocks new and exciting cybersecu-

rity opportunities to the computer vision community—democratizing image-based malware capabilities by enabling researchers and practitioners to evaluate techniques that were previously reported in propriety settings. (Chapter 6).

Open source tools and knowledge repositories.

- We contribute TIGER, the first open-sourced Python toolbox for graph vulnerability and robustness analysis. TIGER contains 22 graph robustness measures with both original and fast approximate versions; 17 failure and attack strategies; 15 heuristic and optimization based defense techniques; and 4 simulation tools (Chapter 3).
- We distill key findings across numerous graph vulnerability and robustness domains in the form of a survey paper, providing researchers access to crucial knowledge by— (1) summarizing recent and classical graph robustness measures; (2) exploring which robustness measures are most applicable to different domains (e.g., social, infrastructure); (3) reviewing attack strategy effectiveness across network topologies; and (4) extensive discussion on selecting defense techniques to mitigate attacks (Chapter 2).

9.2 Impact

- D²M (Chapter 4) has led to major impact to the **Microsoft Defender Advanced Threat Protection product**, inspiring changes to the product’s approach to detect and prevent lateral movement, well known as one of the most challenging areas of post-breach detection.
- TIGER (Chapter 3) has been integrated into the **Nvidia Data Science Teaching Kit** available to educators across the world; and Georgia Tech’s Data and Visual Analytics class with over 1,000 students.
- UNMASK (Chapter 7) helped win a **multi-million dollar** DARPA GARD (Guaranteeing AI Robustness against Deception) grant.
- MALNET-GRAPH (Chapter 5) represents the **worlds largest graph representation learning database**, enabling new research and discoveries into imbalanced classification, explainability and the impact of class hardness.
- MALNET-IMAGE (Chapter 6) is the **worlds largest binary-image database**, unlocking new and unique opportunities to advance the frontiers of vision-based cyber defenses, multi-class imbalanced classification, and interpretable security.
- Our innovations in graph mining, deep learning, cybersecurity and healthcare were recognized and invested in by an **IBM PhD Fellowship**, a **Raytheon Fellowship**,

and an **NSF Graduate Research Fellowship (GRFP)**.

9.3 Future Directions

The research in this thesis unlocks numerous future research directions, and practical applications to extend the development of the robust models, algorithms, and databases discussed in this work.

9.3.1 *Advancing Vision Based Cybersecurity Research*

Research into developing image-based malware detection and classification algorithms has recently surged across industry (e.g., Intel-Microsoft collaboration on Stamina [235], security companies [233, 236]) and academia [237, 238, 239, 240, 241, 249, 250, 251, 252, 253, 254, 255, 256, 247, 257, 258, 245]. However, existing public datasets contain only a handful of classes and thousands of images, and as the field advances, larger and more challenging datasets are needed for the next generation of models. With the release of MALNET-IMAGE in Chapter 6, containing over 1.2 million software images across a hierarchy of 47 types and 696 families, researchers now have access to a critical resource to develop and benchmark advanced image-based malware detection and classification algorithms, previously restricted to a few industry labs and research teams.

Extending Imbalanced Classification into a New and Diverse High-Impact Domain.

While a large body of research has analyzed binary-images in balanced classification settings, only preliminary work has studied malware detection under data imbalance [253] due to the limited number of classes and images available in existing datasets. As a result, it is unknown whether many techniques may generalize to the binary-image domain, and how they will perform in highly imbalanced classification scenarios. We take a first step in studying this by analyzing Figure 6.7, where we can see that classes containing only a few examples typically underperform relative to their more populous counterparts—highlighting the significant challenge of imbalanced classification in the cybersecurity domain. By releasing MALNET-IMAGE, one of the largest naturally imbalanced databases to date, we hope to foster new interest in this important research area, enabling the machine learning community to impact and generalize across domains.

Interpretable Cybersecurity Research

Preliminary research has demonstrated the importance of attention mechanisms in binary-image malware classification, where extracted regions can provide strong indicators to human analysts, helping guide them to suspicious parts of the bytecode for additional analysis [274, 265]. This includes recent research in salience based methods that automatically discover concepts, helping to identify correlated regions of bytecode [276]. Prior to MALNET-IMAGE, researchers were limited to a small number of malicious families and types, hindering their ability to conduct large-scale explainability studies. With MALNET-IMAGE’s nearly 700 classes, researchers can explore a wide variety of malicious software, enabling new breakthroughs and discoveries. For example, researchers might discover that new types of visualization and sense-making techniques are needed to accurately summarize large volumes of binary-image data to enhance security analysts decision making capabilities.

9.3.2 Advancing Graph Representation Learning Research

In Chapter 5, we show that MALNET-GRAPH unlocks new and unique opportunities to advance the frontiers of graph representation learning by enabling research into *imbalanced classification*, *explainability*, and the impact of *class hardness*.

Class Hardness Exploration

Because of MALNET-GRAPH’s large diversity, it is now possible for researchers to explore why certain classes are more challenging to classify than others. For example, Figure 8.4 shows *Malware++Trj* significantly outperforming both *Troj* and *Adsware*, which contain many more examples. This result is surprising, and provides strong impetus for additional research into class hardness, such as: (a) investigating whether existing methods are flexible enough to represent the diverse graph structures; and (b) inviting researchers to study the similarities across class types (e.g. merge *Spr* and *Spyware*). To support further development in this challenging area, we release the raw VirusTotal reports containing up to 70 labels per graph.

Imbalanced Classification Research

The natural world often follows a long-tailed data distribution where only a few classes account for most of the examples [194]. As evidenced in discovery Chapter 5, the long-tail often causes classifiers to perform well on the majority class, but poorly on rare ones. Unfortunately, imbalanced classification research in the graph domain has yet to receive

much attention, largely because no datasets existed to support the research. By releasing MALNET-GRAPH, the largest naturally imbalanced database to date, we hope foster new interest in this important area.

Reconsidering Merits of Simpler Approaches

In Chapter 5, we show that simpler methods can match or outperform more recent and sophisticated techniques, suggesting that current techniques aiming to capture graph topology are not yet well-reflected for non-attributed graphs, echoing results from [179]. More broadly, our discovery demonstrates—for the first time—such phenomenon at the unprecedented scale and diversity offered by MALNET-GRAPH. We believe our results will inspire researchers to reconsider the merits of simpler approaches and classic techniques, and to build on them to reap their benefits.

Enabling Explainable Research

In Figure 8.4 of Chapter 5, we observe that certain representation techniques better capture particular graph types. For example, Feather, GIN and GCN significantly outperforms other methods on *Clicker++Trojan*. This is an interesting result, as it could provide insight into when one technique is preferred over another (e.g., local neighborhood structure, global graph structure, graph motifs). We believe that the wide range of graph topology and substructures contained in MALNET-GRAPH’s nearly 700 classes will enable new explainability research.

9.3.3 Robust Tools and Algorithms

Through careful analysis of the robustness literature, we identify and distill open problems that have strong potential as future research directions.

Guidelines for Selecting & Developing Measures

Comparing robustness measures in a quantitative manner is still an open challenge. While many works have qualitatively remarked on why certain robustness measures are better suited for certain tasks, there has been no formal study outlining desirable characteristics that a robustness measure should contain. By formalizing these desirable properties into a set axioms, future and existing robustness measures could be compared in an independent and quantitative manner, something that is not currently available. We identified 6 desirable robustness properties across the literature that could form the basis for an axiomatic anal-

ysis of robustness measures [25, 26, 47]. Below, we provide the intuition for each axiom, however, each axiom needs to be formalized and (dis)proven for each robustness measure.

1. **Strictly Monotonic.** When an edge is added to a graph the network connectivity is intrinsically enhanced. A robustness measure should account for this increased connectivity by strictly increasing (or decreasing) for each edge added to the graph.
2. **Redundancy.** A critical ability of any robustness measure is to measure redundancy present in the network. This means that if multiple paths between two nodes exist, the proposed measure should be able to account for both the number of paths and their quality (where smaller paths are better).
3. **Disconnected.** Many real-world graphs contain disconnected components; therefore a measure should be able to evaluate a graphs' robustness independent of the number of disconnected components.
4. **Stable.** A robustness measure should change in proportion to the perturbation of the graph structure. For example, if a single edge is added to a graph, we expect that the measure has a proportionally small response.
5. **Consistent.** Given two graphs with same underlying structure, we would expect them to have similar robustness independent of their size.
6. **Scalable.** Large graphs containing millions (or sometimes billions) of nodes and edges are common. A robustness measure should be scalable to large graphs, where we define scalable as an algorithm subquadratic with respect to the number of nodes and edges.
7. **Intuitive.** Ideally, we want robustness measures to have identifiable connections to the underlying graph topology, and for these connections to be conveyable to non-experts in an understandable manner.

Furthering Interpretability

Ideally, robustness measure should have identifiable connections to the underlying graph topology to explain what the robustness score is indicative of. Recent research has explored this in the more general domains of graph connectivity and ranking [143, 144, 145, 146]. Combining visual representations, helpful interactions, and state-of-the-art attribution and feature visualization techniques together into rich user interfaces could lead to major breakthroughs in understanding graph vulnerability and robustness scores.

Studying Robustness in New Domains

The study of graph vulnerability and robustness is still nascent in the areas of physical security [44], cybersecurity [59] and interdependent and dynamic networks [147]. For physical security, [44] studies the vulnerability and robustness of physical sensor placement to maximize perimeter security while minimizing network latency. They find that perimeter security systems frequently map to circular lattices which suffer from a trade-off between robustness and mean path length (i.e., latency). Future work could analyze alternative perimeter system mappings that optimize for both criteria, while exploring alternative definitions of robustness in physical security. With respect to cybersecurity, [59] attempts to calculate the vulnerability and robustness of enterprise networks by modeling lateral attack movement between computers. However, their unique probabilistic robustness measure is dependent on cyber domain knowledge and the running of many simulations. Future cybersecurity robustness analysis could explore the development of robustness measures that are simulation independent, reducing computational costs and the need for explicit domain knowledge. Many real-world networks are often dynamic and contain multiple interdependent sub-networks. While initial work has looked at real-time robustification of interdependent networks from an edge perspective [147], additional work needs to be done to (i) study dynamic graphs, (ii) comprehensively evaluate various attack and defense scenarios, and (iii) develop unique robustness measures that can better account for the nature of inter-dependent and dynamic networks.

Bridging Graph Robustness & Adversarial Machine Learning

From the machine learning perspective, a majority of current graph robustness research focuses on manipulating graph classifiers or embedding mechanisms into mispredicting the label of a graph [148], or the label of each node in the graph [149]. So far, adversarial machine learning research has yet to deeply delve into the more richly defined network robustness objective centered around a networks' ability to continue functioning when damaged or attacked. However, we believe that there are multiple high-impact connections to explore, including: (1) how does a graph's spectral robustness (e.g., spectral gap) correlate to the vulnerability or robustness of downstream tasks such as node and graph classifiers being attacked; and (2) does optimizing a graph's spectral robustness (e.g., adding or rewiring edges) affect an attackers ability to perturb downstream node and graph classification models. By answering these questions, we can uncover new mechanisms to attack and defend networks while gaining insight into fundamental connections between two important and growing fields.

Developing New Open-Source Tools

While TIGER contains numerous tools and techniques for studying the vulnerability and robustness of networks, future work should look at more complex networks e.g., multi-layer networks, directed graphs, weighted graphs; coordinated attack strategies; and more advanced robustness measures that can account for the additional information contained in complex networks.

REFERENCES

- [1] P. McDaniel, J. Launchbury, B. Martin, C. Wang, and H. Kautz, “Artificial intelligence and cyber security: Opportunities and challenges technical workshop summary report,” 2020.
- [2] T. C. of Economic Advisers, “The cost of malicious cyber activity to the u.s. economy,” 2018.
- [3] CrowdStrike, “Blurring the lines between statecraft and tradecraft,” *Global Threat Report*,
- [4] J. Leskovec, J. Kleinberg, and C. Faloutsos, “Graphs over time: Densification laws, shrinking diameters and possible explanations,” in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, 2005, pp. 177–187.
- [5] H. Tong, B. A. Prakash, C. Tsourakakis, T. Eliassi-Rad, C. Faloutsos, and D. H. Chau, “On the vulnerability of large graphs,” in *2010 IEEE International Conference on Data Mining*, IEEE, 2010, pp. 1091–1096.
- [6] A. Hagberg, N. Lemons, A. Kent, and J. Neil, “Connected components and credential hopping in authentication graphs,” in *SITIS*, IEEE, 2014, pp. 416–423.
- [7] S Duckwall and C Campbell, “Hello my name is microsoft and i have a credential problem,” *Blackhat USA 2013 White Papers*, 2013.
- [8] I. You and K. Yim, “Malware obfuscation techniques: A brief survey,” in *2010 International conference on broadband, wireless computing, communication and applications*, IEEE, 2010, pp. 297–300.
- [9] T. Dullien and R. Rolles, “Graph-based comparison of executable objects,” *SSTIC*, 2005.
- [10] V. S. Sathyanarayan, P. Kohli, and B. Bruhadeshwar, “Signature generation and detection of malware families,” in *Australasian Conference on Information Security and Privacy*, Springer, 2008, pp. 336–349.
- [11] B. Gallagher, “Matching structure and semantics: A survey on graph-based pattern matching,” in *AAAI Fall Symposium: Capturing and Using Patterns for Evidence Detection*, 2006, pp. 45–53.

- [12] X. Hu, T.-c. Chiueh, and K. G. Shin, "Large-scale malware indexing using function-call graphs," in *Proceedings of the 16th ACM conference on Computer and communications security*, ACM, 2009, pp. 611–620.
- [13] J. Kinable and O. Kostakis, "Malware classification based on call graph clustering," *Journal in computer virology*, vol. 7, no. 4, pp. 233–245, 2011.
- [14] O. Kostakis, J. Kinable, H. Mahmoudi, and K. Mustonen, "Improved call graph comparison using simulated annealing," in *Proceedings of the 2011 ACM Symposium on Applied Computing*, ACM, 2011, pp. 1516–1523.
- [15] D. Kong and G. Yan, "Discriminant malware distance learning on structural information for automated malware classification," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2013, pp. 1357–1365.
- [16] H. Gascon, F. Yamaguchi, D. Arp, and K. Rieck, "Structural detection of android malware using embedded call graphs," in *Proceedings of the 2013 ACM workshop on Artificial intelligence and security*, ACM, 2013, pp. 45–54.
- [17] H. Jiang, T. Turki, and J. T. Wang, "Dlgraph: Malware detection using deep learning and graph embedding," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, IEEE, 2018, pp. 1029–1033.
- [18] S. Ranveer and S. Hiray, "Comparative analysis of feature extraction methods of malware detection," *International Journal of Computer Applications*, vol. 120, no. 5, 2015.
- [19] B. M. Altevogt, H. R. Colten, *et al.*, *Sleep disorders and sleep deprivation: an unmet public health problem*. National Academies Press, 2006.
- [20] A. A. of Sleep Medicine *et al.*, "Economic burden of undiagnosed sleep apnea in us is nearly \$150 billion per year," *Published on the American Academy of Sleep Medicine's official website, on August*, vol. 8, 2016.
- [21] V. Chvátal, "Tough graphs and hamiltonian circuits," *Discrete Mathematics*, vol. 5, no. 3, pp. 215–228, 1973.
- [22] D. J. Klein and M. Randić, "Resistance distance," *Journal of mathematical chemistry*, vol. 12, no. 1, pp. 81–95, 1993.
- [23] A. Beygelzimer, G. Grinstein, R. Linsker, and I. Rish, "Improving network robustness by edge modification," *Physica A: Statistical Mechanics and its Applications*, vol. 357, no. 3-4, pp. 593–612, 2005.

- [24] M. S. Krishnamoorthy and B. Krishnamurthy, "Fault diameter of interconnection networks," *Computers & Mathematics with Applications*, vol. 13, no. 5-6, pp. 577–582, 1987.
- [25] W. Ellens and R. E. Kooij, "Graph measures and network robustness," *arXiv preprint arXiv:1311.5064*, 2013.
- [26] H. Chan and L. Akoglu, "Optimizing network robustness by edge rewiring: A general framework," *Data Mining and Knowledge Discovery*, vol. 30, no. 5, pp. 1395–1425, 2016.
- [27] A. Yazdani and P. Jeffrey, "Applying network theory to quantify the redundancy and structural robustness of water distribution systems," *Journal of Water Resources Planning and Management*, vol. 138, no. 2, pp. 153–161, 2012.
- [28] O. Lordan, J. M. Sallan, and P. Simo, "Study of the topology and robustness of airline route networks from the complex network approach: A survey and research agenda," *Journal of Transport Geography*, vol. 37, pp. 112–120, 2014.
- [29] G. A. Pagani and M. Aiello, "The power grid as a complex network: A survey," *Physica A: Statistical Mechanics and its Applications*, vol. 392, no. 11, pp. 2688–2700, 2013.
- [30] L. Cuadra, S. Salcedo-Sanz, J. Del Ser, S. Jiménez-Fernández, and Z. W. Geem, "A critical review of robustness in power grids using complex networks concepts," *Energies*, vol. 8, no. 9, pp. 9211–9265, 2015.
- [31] R. Albert, H. Jeong, and A.-L. Barabási, "Error and attack tolerance of complex networks," *nature*, vol. 406, no. 6794, pp. 378–382, 2000.
- [32] M. J. Alenazi and J. P. Sterbenz, "Evaluation and comparison of several graph robustness metrics to improve network resilience," in *2015 7th International Workshop on Reliable Networks Design and Modeling (RNDM)*, IEEE, 2015, pp. 7–13.
- [33] —, "Comprehensive comparison and accuracy of graph metrics in predicting network resilience," in *2015 11th International Conference on the Design of Reliable Communication Networks (DRCN)*, IEEE, 2015, pp. 157–164.
- [34] M. B. Baig and L. Akoglu, "Correlation of node importance measures: An empirical study through graph robustness," in *Proceedings of the 24th International Conference on World Wide Web*, 2015, pp. 275–281.
- [35] J. S. Baras and P. Hovareshti, "Efficient and robust communication topologies for distributed decision making in networked systems," in *Proceedings of the 48th IEEE*

Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference, IEEE, 2009, pp. 3751–3756.

- [36] K. Berdica, “An introduction to road vulnerability: What has been done, is done and should be done,” *Transport policy*, vol. 9, no. 2, pp. 117–127, 2002.
- [37] A. Bernstein, D. Bienstock, D. Hay, M. Uzunoglu, and G. Zussman, “Power grid vulnerability to geographically correlated failures—analysis and control implications,” in *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, IEEE, 2014, pp. 2634–2642.
- [38] A. Bigdeli, A. Tizghadam, and A. Leon-Garcia, “Comparison of network criticality, algebraic connectivity, and other graph metrics,” in *Proceedings of the 1st Annual Workshop on Simplifying Complex Network for Practitioners*, 2009, pp. 1–6.
- [39] A. N. Bishop and I. Shames, “Link operations for slowing the spread of disease in complex networks,” *EPL (Europhysics Letters)*, vol. 95, no. 1, p. 18 005, 2011.
- [40] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, and D.-U. Hwang, “Complex networks: Structure and dynamics,” *Physics reports*, vol. 424, no. 4-5, pp. 175–308, 2006.
- [41] S. P. Borgatti, K. M. Carley, and D. Krackhardt, “On the robustness of centrality measures under conditions of imperfect data,” *Social networks*, vol. 28, no. 2, pp. 124–136, 2006.
- [42] L. Briesemeister, P. Lincoln, and P. Porras, “Epidemic profiles and defense of scale-free networks,” in *Proceedings of the 2003 ACM workshop on Rapid malware*, 2003, pp. 67–75.
- [43] S. V. Buldyrev, R. Parshani, G. Paul, H. E. Stanley, and S. Havlin, “Catastrophic cascade of failures in interdependent networks,” *Nature*, vol. 464, no. 7291, pp. 1025–1028, 2010.
- [44] R. Byrne, J. Feddema, and C. Abdallah, “Algebraic connectivity and graph robustness,” *SANDIA Report*, vol. 87185, pp. 1–34, 2005.
- [45] J. Caballero, T. Kampouris, D. Song, and J. Wang, “Would diversity really increase the robustness of the routing infrastructure against software defects?” In *NDSS*, Citeseer, 2008.
- [46] D. S. Callaway, M. E. Newman, S. H. Strogatz, and D. J. Watts, “Network robustness and fragility: Percolation on random graphs,” *Physical review letters*, vol. 85, no. 25, p. 5468, 2000.

- [47] H. Chan, L. Akoglu, and H. Tong, “Make it or break it: Manipulating robustness in large networks,” in *Proceedings of the 2014 SIAM International Conference on Data Mining*, SIAM, 2014, pp. 325–333.
- [48] D. Chakrabarti, Y. Wang, C. Wang, J. Leskovec, and C. Faloutsos, “Epidemic thresholds in real networks,” *ACM Transactions on Information and System Security (TISSEC)*, vol. 10, no. 4, pp. 1–26, 2008.
- [49] C. Chen, H. Tong, B. A. Prakash, C. E. Tsourakakis, T. Eliassi-Rad, C. Faloutsos, and D. H. Chau, “Node immunization on large graphs: Theory and algorithms,” *TKDE*, vol. 28, no. 1, pp. 113–126, 2015.
- [50] C. Chen, J. He, N. Bliss, and H. Tong, “On the connectivity of multi-layered networks: Models, measures and optimal control,” in *2015 IEEE International Conference on Data Mining*, IEEE, 2015, pp. 715–720.
- [51] C. Chen, H. Tong, B. A. Prakash, T. Eliassi-Rad, M. Faloutsos, and C. Faloutsos, “Eigen-optimization on large graphs by edge manipulation,” *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 10, no. 4, pp. 1–30, 2016.
- [52] P. Crucitti, V. Latora, and M. Marchiori, “Model for cascading failures in complex networks,” *Physical Review E*, vol. 69, no. 4, p. 045 104, 2004.
- [53] A. H. Dekker, “Simulating network robustness for critical infrastructure networks,” in *ACM International Conference Proceeding Series*, Citeseer, vol. 102, 2005, pp. 59–67.
- [54] S. Derrible and C. Kennedy, “The complexity and robustness of metro networks,” *Physica A: Statistical Mechanics and its Applications*, vol. 389, no. 17, pp. 3678–3691, 2010.
- [55] Y. Duan and F. Lu, “Robustness of city road networks at different granularities,” *Physica A: Statistical Mechanics and its Applications*, vol. 411, pp. 21–34, 2014.
- [56] W. Ellens, F. Spieksma, P. Van Mieghem, A. Jamakovic, and R. Kooij, “Effective graph resistance,” *Linear algebra and its applications*, vol. 435, no. 10, pp. 2491–2506, 2011.
- [57] E. Estrada, “Network robustness to targeted attacks. the interplay of expansibility and degree distribution,” *The European Physical Journal B-Condensed Matter and Complex Systems*, vol. 52, no. 4, pp. 563–574, 2006.
- [58] —, “Spectral scaling and good expansion properties in complex networks,” *EPL (Europhysics Letters)*, vol. 73, no. 4, p. 649, 2006.

- [59] S. Freitas, A. Wicker, D. H. Chau, and J. Neil, “D2m: Dynamic defense and modeling of adversarial movement in networks,” *SDM*, pp. 541–549, 2020.
- [60] S. Freitas and D. H. Chau, “Evaluating graph vulnerability and robustness using tiger,” *arXiv preprint arXiv:2006.05648*, 2020.
- [61] J. Gao, S. V. Buldyrev, S. Havlin, and H. E. Stanley, “Robustness of a network of networks,” *Physical Review Letters*, vol. 107, no. 19, p. 195 701, 2011.
- [62] A. Ghosh, S. Boyd, and A. Saberi, “Minimizing effective resistance of a graph,” *SIAM review*, vol. 50, no. 1, pp. 37–66, 2008.
- [63] P. Holme, B. J. Kim, C. N. Yoon, and S. K. Han, “Attack vulnerability of complex networks,” *Physical review E*, vol. 65, no. 5, p. 056 109, 2002.
- [64] Å. J. Holmgren, “Using graph models to analyze the vulnerability of electric power networks,” *Risk analysis*, vol. 26, no. 4, pp. 955–969, 2006.
- [65] A. Jamakovic and S. Uhlig, “On the relationship between the algebraic connectivity and graph’s robustness to node and link failures,” in *2007 Next Generation Internet Networks*, IEEE, 2007, pp. 96–102.
- [66] E. B. Khalil, B. Dilkina, and L. Song, “Scalable diffusion-aware optimization of network topology,” in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 1226–1235.
- [67] R. Kinney, P. Crucitti, R. Albert, and V. Latora, “Modeling cascading failures in the north american power grid,” *The European Physical Journal B-Condensed Matter and Complex Systems*, vol. 46, no. 1, pp. 101–107, 2005.
- [68] G. W. Klau and R. Weiskircher, “Robustness and resilience,” in *Network analysis*, Springer, 2005, pp. 417–437.
- [69] V. Latora and M. Marchiori, “Vulnerability and protection of infrastructure networks,” *Physical Review E*, vol. 71, no. 1, p. 015 103, 2005.
- [70] L. T. Le, T. Eliassi-Rad, and H. Tong, “Met: A fast algorithm for minimizing propagation in large graphs with small eigen-gaps,” in *Proceedings of the 2015 SIAM International Conference on Data Mining*, SIAM, 2015, pp. 694–702.
- [71] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance, “Cost-effective outbreak detection in networks,” in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2007, pp. 420–429.

- [72] J. Liu, M. Zhou, S. Wang, and P. Liu, “A comparative study of network robustness measures,” *Frontiers of Computer Science*, vol. 11, no. 4, pp. 568–584, 2017.
- [73] Z.-M. Lu and X.-F. Li, “Attack vulnerability of network controllability,” *PloS one*, vol. 11, no. 9, e0162289, 2016.
- [74] F. D. Malliaros, V. Megalooikonomou, and C. Faloutsos, “Fast robustness estimation in large social graphs: Communities and anomaly detection,” in *Proceedings of the 2012 SIAM International Conference on Data Mining*, SIAM, 2012, pp. 942–953.
- [75] J. L. Marzo, E. Calle, S. G. Cosgaya, D. Rueda, and A. Mañosa, “On selecting the relevant metrics of network robustness,” in *2018 10th International Workshop on Resilient Networks Design and Modeling (RNDM)*, IEEE, 2018, pp. 1–7.
- [76] L.-G. Mattsson and E. Jenelius, “Vulnerability and resilience of transport systems—a discussion of recent research,” *Transportation Research Part A: Policy and Practice*, vol. 81, pp. 16–34, 2015.
- [77] P. Van Mieghem, D. Stevanović, F. Kuipers, C. Li, R. Van De Bovenkamp, D. Liu, and H. Wang, “Decreasing the spectral radius of a graph by link removals,” *Physical Review E*, vol. 84, no. 1, p. 016 101, 2011.
- [78] A. Milanese, J. Sun, and T. Nishikawa, “Approximating spectral impact of structural perturbations in large networks,” *Physical Review E*, vol. 81, no. 4, p. 046 112, 2010.
- [79] A. E. Motter and Y.-C. Lai, “Cascade-based attacks on complex networks,” *Physical Review E*, vol. 66, no. 6, p. 065 102, 2002.
- [80] A. Di Nardo, C. Giudicianni, R. Greco, M. Herrera, and G. F. Santonastaso, “Applications of graph spectral techniques to water distribution network management,” *Water*, vol. 10, no. 1, p. 45, 2018.
- [81] D. T. Nguyen, Y. Shen, and M. T. Thai, “Detecting critical nodes in interdependent power networks for vulnerability assessment,” *IEEE Transactions on Smart Grid*, vol. 4, no. 1, pp. 151–159, 2013.
- [82] M. Parandehgheibi and E. Modiano, “Robustness of interdependent networks: The case of communication networks and the power grid,” in *2013 IEEE Global Communications Conference (GLOBECOM)*, IEEE, 2013, pp. 2164–2169.
- [83] R. Parshani, S. V. Buldyrev, and S. Havlin, “Interdependent networks: Reducing the coupling strength leads to a change from a first to second order percolation transition,” *Physical review letters*, vol. 105, no. 4, p. 048 701, 2010.

- [84] G. Paul, T Tanizawa, S. Havlin, and H. E. Stanley, "Optimization of robustness of complex networks," *The European Physical Journal B*, vol. 38, no. 2, pp. 187–191, 2004.
- [85] B. A. Prakash, H. Tong, N. Valler, M. Faloutsos, and C. Faloutsos, "Virus propagation on time-varying networks: Theory and immunization algorithms," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, 2010, pp. 99–114.
- [86] B. A. Prakash, D. Chakrabarti, N. C. Valler, M. Faloutsos, and C. Faloutsos, "Threshold conditions for arbitrary cascade models on arbitrary networks," *Knowledge and information systems*, vol. 33, no. 3, pp. 549–575, 2012.
- [87] B. A. Prakash, L. Adamic, T. Iwashyna, H. Tong, and C. Faloutsos, "Fractional immunization in networks," in *Proceedings of the 2013 SIAM International Conference on Data Mining*, SIAM, 2013, pp. 659–667.
- [88] D. F. Rueda, E. Calle, and J. L. Marzo, "Robustness comparison of 15 real telecommunication networks: Structural and centrality measurements," *Journal of Network and Systems Management*, vol. 25, no. 2, pp. 269–289, 2017.
- [89] S. Saha, A. Adiga, B. A. Prakash, and A. Vullikanti, "Approximation algorithms for reducing the spectral radius to control epidemic spread," in *SDM'15*, SIAM, 2015, pp. 568–576.
- [90] C. M. Schneider, A. A. Moreira, J. S. Andrade, S. Havlin, and H. J. Herrmann, "Mitigation of malicious attacks on networks," *Proceedings of the National Academy of Sciences*, vol. 108, no. 10, pp. 3838–3841, 2011.
- [91] C. M. Schneider, T. Mihaljev, S. Havlin, and H. J. Herrmann, "Suppressing epidemics with a limited amount of immunization units," *Physical Review E*, vol. 84, no. 6, p. 061 911, 2011.
- [92] J. Shao, S. V. Buldyrev, S. Havlin, and H. E. Stanley, "Cascade of failures in coupled network systems with multiple support-dependence relations," *Physical Review E*, vol. 83, no. 3, p. 036 116, 2011.
- [93] B. Shargel, H. Sayama, I. R. Epstein, and Y. Bar-Yam, "Optimization of robustness and connectivity in complex networks," *Physical review letters*, vol. 90, no. 6, p. 068 701, 2003.
- [94] A. Sydney, C. Scoglio, P. Schumm, and R. Kooij, "Elasticity: Topological characterization of robustness in complex networks," *arXiv preprint arXiv:0811.4040*, 2008.

- [95] G. Tanaka, K. Morino, and K. Aihara, “Dynamical robustness in complex networks: The crucial role of low-degree nodes,” *Nature*, vol. 2, no. 1, pp. 1–6, 2012.
- [96] H. Tong, B. A. Prakash, T. Eliassi-Rad, M. Faloutsos, and C. Faloutsos, “Gelling, and melting, large graphs by edge manipulation,” in *Proceedings of the 21st ACM international conference on Information and knowledge management*, 2012, pp. 245–254.
- [97] L. Torres, K. S. Chan, H. Tong, and T. Eliassi-Rad, “Node immunization with non-backtracking eigenvalues,” *arXiv preprint arXiv:2002.12309*, 2020.
- [98] S. Trajanovski, J. Martín-Hernández, W. Winterbach, and P. Van Mieghem, “Robustness envelopes of networks,” *Journal of Complex Networks*, vol. 1, no. 1, pp. 44–62, 2013.
- [99] A. Vespignani, “The fragility of interdependency,” *Nature*, vol. 464, no. 7291, pp. 984–985, 2010.
- [100] J. Wang, L. Rong, L. Zhang, and Z. Zhang, “Attack vulnerability of scale-free networks due to cascading failures,” *Physica A: Statistical Mechanics and its Applications*, vol. 387, no. 26, pp. 6671–6678, 2008.
- [101] X. Wang, E. Pournaras, R. E. Kooij, and P. Van Mieghem, “Improving robustness of complex networks via the effective graph resistance,” *The European Physical Journal B*, vol. 87, no. 9, pp. 1–12, 2014.
- [102] D. J. Watts and S. H. Strogatz, “Collective dynamics of ‘small-world’ networks,” *nature*, vol. 393, no. 6684, pp. 440–442, 1998.
- [103] W. Jun, M. Barahona, T. Yue-Jin, and D. Hong-Zhong, “Natural connectivity of complex networks,” *Chinese physics letters*, vol. 27, no. 7, p. 078 902, 2010.
- [104] J. Wu, M. Barahona, Y.-J. Tan, and H.-Z. Deng, “Spectral measure of structural robustness in complex networks,” *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 41, no. 6, pp. 1244–1252, 2011.
- [105] Y. Xia, J. Fan, and D. Hill, “Cascading failure in watts–strogatz small-world networks,” *Physica A: Statistical Mechanics and its Applications*, vol. 389, no. 6, pp. 1281–1285, 2010.
- [106] Y. Yang, Z. Li, Y. Chen, X. Zhang, and S. Wang, “Improving the robustness of complex networks with preserving community structure,” *PloS one*, vol. 10, no. 2, e0116551, 2015.

- [107] A Yazdani and P Jeffrey, “Robustness and vulnerability analysis of water distribution networks using graph theoretic and complex network principles,” in *Water Distribution Systems Analysis 2010*, 2010, pp. 933–945.
- [108] A. Yazdani and P. Jeffrey, “Complex network analysis of water distribution systems,” *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 21, no. 1, p. 016 111, 2011.
- [109] A. Zeng and W. Liu, “Enhancing network robustness against malicious attacks,” *Physical Review E*, vol. 85, no. 6, p. 066 130, 2012.
- [110] L. Zhao, K. Park, and Y.-C. Lai, “Attack vulnerability of scale-free networks due to cascading breakdown,” *Physical review E*, vol. 70, no. 3, p. 035 101, 2004.
- [111] D. Zhao, L. Wang, S. Li, Z. Wang, L. Wang, and B. Gao, “Immunization of epidemics in multiplex networks,” *PloS one*, vol. 9, no. 11, e112018, 2014.
- [112] H. A. Jung, “On a class of posets and the corresponding comparability graphs,” *Journal of Combinatorial Theory, Series B*, vol. 24, no. 2, pp. 125–133, 1978.
- [113] M. Cozzens, D. Moazzami, and S. Stueckle, “The tenacity of a graph,” 1995.
- [114] C. Barefoot, R. Entringer, and H. Swart, “Integrity of trees and powers of cycles,” *Congr. Numer*, vol. 58, pp. 103–114, 1987.
- [115] B. Mohar, “Isoperimetric numbers of graphs,” *Journal of combinatorial theory, Series B*, vol. 47, no. 3, pp. 274–291, 1989.
- [116] A.-H. Esfahanian, “Connectivity algorithms,” in *Topics in structural graph theory*, Cambridge University Press, 2013, pp. 268–281.
- [117] D. W. Matula, “Determining edge connectivity in $O(nm)$,” in *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, IEEE, 1987, pp. 249–251.
- [118] H. Whitney, “Congruent graphs and the connectivity of graphs,” in *Hassler Whitney Collected Papers*, Springer, 1992, pp. 61–79.
- [119] R. W. Floyd, “Algorithm 97: Shortest path,” *Commun. ACM*, vol. 5, no. 6, p. 345, Jun. 1962.
- [120] L. C. Freeman, “A set of measures of centrality based on betweenness,” *Sociometry*, pp. 35–41, 1977.
- [121] U. Brandes, “A faster algorithm for betweenness centrality,” *Journal of mathematical sociology*, vol. 25, no. 2, pp. 163–177, 2001.

- [122] O. Green and D. A. Bader, “Faster clustering coefficient using vertex covers,” in *2013 International Conference on Social Computing*, IEEE, 2013, pp. 321–330.
- [123] S. K. Butler, “Eigenvalues and structures of graphs,” Ph.D. dissertation, UC San Diego, 2008.
- [124] Y. Wang, D. Chakrabarti, C. Wang, and C. Faloutsos, “Epidemic spreading in real networks: An eigenvalue viewpoint,” in *22nd International Symposium on Reliable Distributed Systems, 2003. Proceedings.*, IEEE, 2003, pp. 25–34.
- [125] C. Chen and H. Tong, “Fast eigen-functions tracking on dynamic graphs,” in *Proceedings of the 2015 SIAM International Conference on Data Mining*, SIAM, 2015, pp. 559–567.
- [126] E. Estrada and J. A. Rodríguez-Velázquez, “Subgraph centrality in complex networks,” *Physical Review E*, vol. 71, no. 5, p. 056 103, 2005.
- [127] S. Hoory, N. Linial, and A. Wigderson, “Expander graphs and their applications,” *Bulletin of the American Mathematical Society*, vol. 43, no. 4, pp. 439–561, 2006.
- [128] E. Estrada and J. A. Rodríguez-Velázquez, “Spectral measures of bipartivity in complex networks,” *Physical Review E*, vol. 72, no. 4, p. 046 105, 2005.
- [129] L. Wu, P.-Y. Chen, I. E.-H. Yen, F. Xu, Y. Xia, and C. Aggarwal, “Scalable spectral clustering using random binning features,” in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 2506–2515.
- [130] M. Fiedler, “Algebraic connectivity of graphs,” *Czechoslovak mathematical journal*, vol. 23, no. 2, pp. 298–305, 1973.
- [131] R. B. Lehoucq, D. C. Sorensen, and C. Yang, *ARPACK users’ guide: solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods*. Siam, 1998, vol. 6.
- [132] G. Weichenberg, V. W. Chan, and M. Médard, “High-reliability topological architectures for networks under stress,” *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 9, pp. 1830–1845, 2004.
- [133] F. Buekenhout and M. Parker, “The number of nets of the regular convex polytopes in dimension ≥ 4 ,” *Discrete mathematics*, vol. 186, no. 1-3, pp. 69–94, 1998.
- [134] S. Tsironis, M. Sozio, M. Vazirgiannis, and L. Polte, “Accurate spectral clustering for community detection in mapreduce,” in *Advances in Neural Information Processing Systems (NIPS) Workshops*, Citeseer, 2013.

- [135] A. Di Nardo, C. Giudicianni, R. Greco, M. Herrera, and G. F. Santonastaso, “Applications of graph spectral techniques to water distribution network management,” *Water*, vol. 10, no. 1, p. 45, 2018.
- [136] P. Erdős and A. Rényi, “On the evolution of random graphs,” *Publ. Math. Inst. Hung. Acad. Sci.*, vol. 5, no. 1, pp. 17–60, 1960.
- [137] D. Chakrabarti and C. Faloutsos, “Graph mining: Laws, generators, and algorithms,” *ACM computing surveys (CSUR)*, vol. 38, no. 1, 2–es, 2006.
- [138] A.-L. Barabási and R. Albert, “Emergence of scaling in random networks,” *science*, vol. 286, no. 5439, pp. 509–512, 1999.
- [139] P. Holme and B. J. Kim, “Growing scale-free networks with tunable clustering,” *Physical review E*, vol. 65, no. 2, p. 026 107, 2002.
- [140] L. Page, S. Brin, R. Motwani, and T. Winograd, “The pagerank citation ranking: Bringing order to the web.,” Stanford InfoLab, Tech. Rep., 1999.
- [141] T. Opsahl, F. Agneessens, and J. Skvoretz, “Node centrality in weighted networks: Generalizing degree and shortest paths,” *Social networks*, vol. 32, no. 3, pp. 245–251, 2010.
- [142] L. Katz, “A new status index derived from sociometric analysis,” *Psychometrika*, vol. 18, no. 1, pp. 39–43, 1953.
- [143] J. Kang, M. Wang, N. Cao, Y. Xia, W. Fan, and H. Tong, “Aurora: Auditing pagerank on large graphs,” in *2018 IEEE International Conference on Big Data (Big Data)*, IEEE, 2018, pp. 713–722.
- [144] M. Wang, J. Kang, C. Nan, Y. Xia, W. Fan, and H. Tong, “Graph ranking auditing: Problem definition and fast solutions,” *IEEE Transactions on Knowledge and Data Engineering*, 2020.
- [145] J. Kang, S. Freitas, H. Yu, Y. Xia, N. Cao, and H. Tong, “X-rank: Explainable ranking in complex multi-layered networks,” in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, 2018, pp. 1959–1962.
- [146] T. Xie, Y. Ma, H. Tong, M. T. Thai, and R. Maciejewski, “Auditing the sensitivity of graph-based ranking with visual analytics,” *IEEE Transactions on Visualization and Computer Graphics*, 2020.

- [147] Z. Chen, H. Tong, and L. Ying, “Realtime robustification of interdependent networks under cascading attacks,” in *2018 IEEE International Conference on Big Data (Big Data)*, IEEE, 2018, pp. 1347–1356.
- [148] H. Dai, H. Li, T. Tian, X. Huang, L. Wang, J. Zhu, and L. Song, “Adversarial attack on graph structured data,” *arXiv preprint arXiv:1806.02371*, 2018.
- [149] D. Zügner, A. Akbarnejad, and S. Günnemann, “Adversarial attacks on neural networks for graph data,” in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 2847–2856.
- [150] A. Hagberg, P. Swart, and D. S Chult, “Exploring network structure, dynamics, and function using networkx,” Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2008.
- [151] A. Azizi, C. Montalvo, B. Espinoza, Y. Kang, and C. Castillo-Chavez, “Epidemics on networks: Reducing disease transmission using health emergency declarations and peer communication,” *Infectious Disease Modelling*, vol. 5, pp. 12–22, 2020.
- [152] G. Rossetti, L. Milli, S. Rinzivillo, A. Sîrbu, D. Pedreschi, and F. Giannotti, “Ndlb: A python library to model and analyze diffusion processes over complex networks,” *International Journal of Data Science and Analytics*, vol. 5, no. 1, pp. 61–79, 2018.
- [153] K. A. Klise, R. Murray, and T. Haxton, “An overview of the water network tool for resilience (wntr).,” Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), Tech. Rep., 2018.
- [154] M. Korkali, J. G. Veneman, B. F. Tivnan, J. P. Bagrow, and P. D. Hines, “Reducing cascading failure risk by increasing infrastructure network interdependence,” *Scientific reports*, vol. 7, p. 44 499, 2017.
- [155] S. Freitas, H. Tong, N. Cao, and Y. Xia, “Rapid analysis of network connectivity,” in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 2017, pp. 2463–2466.
- [156] M. Bastian, S. Heymann, and M. Jacomy, “Gephi: An open source software for exploring and manipulating networks,” in *Third international AAAI conference on weblogs and social media*, 2009.
- [157] U. Brandes and C. Pich, “Centrality estimation in large networks,” *International Journal of Bifurcation and Chaos*, vol. 17, no. 07, pp. 2303–2318, 2007.
- [158] E. Hernadez, S. Hoagland, and L. Ormsbee, “Water distribution database for research applications,” in *World Environmental and Water Resources Congress 2016*, 2016, pp. 465–474.

- [159] W. O. Kermack and A. G. McKendrick, "A contribution to the mathematical theory of epidemics," *Proceedings of the royal society of london. Series A, Containing papers of a mathematical and physical character*, vol. 115, no. 772, pp. 700–721, 1927.
- [160] I. Hernandez-Fajardo and L. Dueñas-Osorio, "Probabilistic study of cascading failures in complex interdependent lifeline systems," *Reliability Engineering & System Safety*, vol. 111, pp. 260–272, 2013.
- [161] M. N. Banu and S. M. Banu, "A comprehensive study of phishing attacks," *IJCSIT*, vol. 4, no. 6, pp. 783–786, 2013.
- [162] R. Lefferts, *Gartner names microsoft a leader in 2019 endpoint protection platforms magic quadrant*.
- [163] Q. Liu, J. W. Stokes, R. Mead, T. Burrell, I. Hellen, J. Lambert, A. Marochko, and W. Cui, "Latte: Large-scale lateral movement detection," in *MILCOM*, IEEE, 2018, pp. 1–6.
- [164] J. Neil, C. Hash, A. Brugh, M. Fisk, and C. B. Storlie, "Scan statistics for the online detection of locally anomalous subgraphs," *Technometrics*, 2013.
- [165] M. A. Nouredine, A. Fawaz, W. H. Sanders, and T. Başar, "A game-theoretic approach to respond to attacker lateral movement," in *GameSec*, Springer, 2016.
- [166] A. Fawaz, A. Bohara, C. Cheh, and W. H. Sanders, "Lateral movement detection using distributed data fusion," in *SRDS*, IEEE, 2016, pp. 21–30.
- [167] A. D. Kent, L. M. Liebrock, and J. C. Neil, "Authentication graphs: Analyzing user behavior within an enterprise network," *Computers & Security*, vol. 48, 2015.
- [168] O. Sheyner and J. Wing, "Tools for generating and analyzing attack graphs," in *International Symposium on Formal Methods for Components and Objects*.
- [169] P. Ammann, D. Wijesekera, and S. Kaushik, "Scalable, graph-based network vulnerability analysis," in *CCS*, ACM, 2002, pp. 217–224.
- [170] S. Jha, O. Sheyner, and J. Wing, "Two formal analyses of attack graphs," in *Proceedings 15th IEEE Computer Security Foundations Workshop. CSFW-15*.
- [171] "The ntlm authentication protocol and security support provider," Tech. Rep., 2006.
- [172] B. C. Neuman and T. Ts'o, "Kerberos: An authentication service for computer networks," *IEEE Communications magazine*, vol. 32, no. 9, pp. 33–38, 1994.

- [173] M Soria-Machado, D Abolins, C Boldea, and K Socha, “Detecting lateral movements in windows infrastructure,” *CERT-EU Security Whitepaper 17-002*,
- [174] J. Mulder, *Mimikatz overview, defenses and detection*, 2016.
- [175] J. Sexton, C. Storlie, and J. Neil, “Attack chain detection,” *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 8, no. 5-6, pp. 353–363, 2015.
- [176] V. L. Le, I. Welch, X. Gao, and P. Komisarczuk, “Anatomy of drive-by download attack,” in *ACSW-AISC*, Australian Computer Society, Inc., 2013, pp. 49–58.
- [177] M. E. Newman, “Mathematics of networks,” *The new Palgrave dictionary of economics*, pp. 1–8, 2016.
- [178] A. D. Kent, “Cybersecurity Data Sources for Dynamic Network Research,” in *Dynamic Networks in Cybersecurity*, Imperial College Press, Jun. 2015.
- [179] C. Cai and Y. Wang, “A simple yet effective baseline for non-attributed graph classification,” *ICLR*, 2019.
- [180] F. Errica, M. Podda, D. Bacciu, and A. Micheli, “A fair comparison of graph neural networks for graph classification,” *arXiv preprint arXiv:1912.09893*, 2019.
- [181] T. Schulz and P. Welke, “On the necessity of graph kernel baselines,” in *ECML-PKDD, GEM workshop*, 2019.
- [182] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, “Pitfalls of graph neural network evaluation,” *arXiv preprint arXiv:1811.05868*, 2018.
- [183] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel, “Protein function prediction via graph kernels,” *Bioinformatics*, vol. 21, 2005.
- [184] P. Yanardag and S. Vishwanathan, “Deep graph kernels,” in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, 2015, pp. 1365–1374.
- [185] Y. LeCun, “The mnist database of handwritten digits,” <http://yann.lecun.com/exd-b/mnist/>, 1998.
- [186] M. Hurier, G. Suarez-Tangil, S. K. Dash, T. F. Bissyandé, Y. Le Traon, J. Klein, and L. Cavallaro, “Euphony: Harmonious unification of cacophonous anti-virus vendor labels for android malware,” in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, IEEE, 2017, pp. 425–435.

- [187] V. Total, “VirusTotal-free online virus, malware and url scanner,” *Online*: <https://www.virustotal.com/en>, 2012.
- [188] M. Hurier, K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, “On the lack of consensus in anti-virus decisions: Metrics and insights on building ground truths of android malware,” in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer, 2016, pp. 142–162.
- [189] A. Kantchelian, M. C. Tschantz, S. Afroz, B. Miller, V. Shankar, R. Bachwani, A. D. Joseph, and J. D. Tygar, “Better malware ground truth: Techniques for weighting anti-virus vendor labels,” in *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security*, 2015, pp. 45–56.
- [190] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “Smote: Synthetic minority over-sampling technique,” *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.
- [191] M. Peng, Q. Zhang, X. Xing, T. Gui, X. Huang, Y.-G. Jiang, K. Ding, and Z. Chen, “Trainable undersampling for class-imbalance learning,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 4707–4714.
- [192] K. Cao, C. Wei, A. Gaidon, N. Arechiga, and T. Ma, “Learning imbalanced datasets with label-distribution-aware margin loss,” in *Advances in Neural Information Processing Systems*, 2019, pp. 1565–1576.
- [193] Y. Cui, M. Jia, T.-Y. Lin, Y. Song, and S. Belongie, “Class-balanced loss based on effective number of samples,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9268–9277.
- [194] R. Duggal, S. Freitas, S. Dhamnani, D. Horng, J. Sun, *et al.*, “Elf: An early-exiting framework for long-tailed classification,” *arXiv:2006.11979*, 2020.
- [195] B. Ramsundar, S. Kearnes, P. Riley, D. Webster, D. Konerding, and V. Pande, “Massively multitask networks for drug discovery,” *arXiv preprint arXiv:1502.02072*, 2015.
- [196] S. G. Rohrer and K. Baumann, “Maximum unbiased validation (muv) data sets for virtual screening based on pubchem bioactivity data,” *Journal of chemical information and modeling*, vol. 49, no. 2, pp. 169–184, 2009.
- [197] X. Yan, H. Cheng, J. Han, and P. S. Yu, “Mining significant graph patterns by leap search,” in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, 2008, pp. 433–444.

- [198] Z. Wu, B. Ramsundar, E. N. Feinberg, J. Gomes, C. Geniesse, A. S. Pappu, K. Leswing, and V. Pande, "Moleculenet: A benchmark for molecular machine learning," *Chemical science*, vol. 9, no. 2, pp. 513–530, 2018.
- [199] N. Wale, I. A. Watson, and G. Karypis, "Comparison of descriptor spaces for chemical compound retrieval and classification," *Knowledge and Information Systems*, vol. 14, no. 3, pp. 347–375, 2008.
- [200] H. Toivonen, A. Srinivasan, R. D. King, S. Kramer, and C. Helma, "Statistical evaluation of the predictive toxicology challenge 2000–2001," *Bioinformatics*, vol. 19, no. 10, pp. 1183–1193, 2003.
- [201] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch, "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity," *Journal of medicinal chemistry*, vol. 34, no. 2, pp. 786–797, 1991.
- [202] K. Riesen and H. Bunke, "Iam graph database repository for graph based pattern recognition and machine learning," in *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, Springer, 2008, pp. 287–297.
- [203] M. Neumann, P. Moreno, L. Antanas, R. Garnett, and K. Kersting, "Graph kernels for object category prediction in task-dependent robot grasping," in *Online Proceedings of the Eleventh Workshop on Mining and Learning with Graphs*, 2013, pp. 0–6.
- [204] P. D. Dobson and A. J. Doig, "Distinguishing enzyme structures from non-enzymes without alignments," *Journal of molecular biology*, vol. 330, no. 4, pp. 771–783, 2003.
- [205] B. Rozemberczki, O. Kiss, and R. Sarkar, "An api oriented open-source python framework for unsupervised learning on graphs," *CIKM*, 2020.
- [206] "Aids antiviral screen," URL <http://dtp.nci.nih.gov/docs/aids/aids-data.html>, 2004.
- [207] Y. Wang, J. Xiao, T. O. Suzek, J. Zhang, J. Wang, Z. Zhou, L. Han, K. Karapetyan, S. Dracheva, B. A. Shoemaker, *et al.*, "Pubchem's bioassay database," *Nucleic acids research*, vol. 40, no. D1, pp. D400–D412, 2012.
- [208] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou, "Deep ground truth analysis of current android malware," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer, 2017, pp. 252–276.

- [209] B. Popper, *Google announces over 2 billion monthly active devices on android*, May 2017.
- [210] L. Li, J. Gao, M. Hurier, P. Kong, T. F. Bissyandé, A. Bartel, J. Klein, and Y. L. Traon, “Androzoo++: Collecting millions of android apps and their metadata for the research community,” *arXiv preprint arXiv:1709.05281*, 2017.
- [211] “Nokia threat intelligence report,” *Network Security*, 2018.
- [212] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, “Androzoo: Collecting millions of android apps for the research community,” in *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, IEEE, 2016, pp. 468–471.
- [213] A. Desnos and G. Gueguen, “Android: From reversing to decompilation,” *Proc. of Black Hat Abu Dhabi*, pp. 77–101, 2011.
- [214] R. Duggal, S. Freitas, C. Xiao, D. H. Chau, and J. Sun, “Rest: Robust and efficient neural networks for sleep monitoring in the wild,” in *Proceedings of The Web Conference 2020*, 2020, pp. 1704–1714.
- [215] M. Fey and J. E. Lenssen, “Fast graph representation learning with pytorch geometric,” *arXiv preprint arXiv:1903.02428*, 2019.
- [216] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” in *International Conference on Learning Representations (ICLR)*, 2017.
- [217] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, “How powerful are graph neural networks?” In *International Conference on Learning Representations*, 2019.
- [218] B. Rozemberczki and R. Sarkar, “Characteristic functions on graphs: Birds of a feather, from statistical descriptors to parametric models,” in *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20)*, ACM, 2020.
- [219] A. Tsitsulin, M. Munkhoeva, and B. Perozzi, “Just slaa when you approximate: Accurate spectral distances for web-scale graphs,” in *WWW '20*, Taipei, Taiwan, 2020.
- [220] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.
- [221] F. Wu, T. Zhang, A. H. d. Souza Jr, C. Fifty, T. Yu, and K. Q. Weinberger, “Simplifying graph convolutional networks,” *arXiv preprint arXiv:1902.07153*, 2019.

- [222] P.-Y. Chen, L. Wu, S. Liu, and I. Rajapakse, “Fast incremental von neumann graph entropy computation: Theory, algorithm, and applications,” in *International Conference on Machine Learning*, 2019, pp. 1091–1101.
- [223] A. Tsitsulin, D. Mottin, P. Karras, A. Bronstein, and E. Müller, “Netlsd: Hearing the shape of a graph,” in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 2347–2356.
- [224] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, “Graph kernels,” *The Journal of Machine Learning Research*, vol. 11, pp. 1201–1242, 2010.
- [225] K. M. Borgwardt and H.-P. Kriegel, “Shortest-path kernels on graphs,” in *Fifth IEEE international conference on data mining (ICDM’05)*, IEEE, 2005, 8–pp.
- [226] F. Johansson, V. Jethava, D. Dubhashi, and C. Bhattacharyya, “Global graph kernels using geometric embeddings,” in *Proceedings of the 31st International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, 2014.
- [227] F. Gao, G. Wolf, and M. Hirn, “Geometric scattering for graph data analysis,” in *International Conference on Machine Learning*, 2019, pp. 2122–2131.
- [228] N. de Lara and E. Pineau, “A simple baseline algorithm for graph classification,” *arXiv preprint arXiv:1810.09155*, 2018.
- [229] A. Galland and M. Lelarge, “Invariant embedding for graph classification,” in *ICML 2019 Workshop on Learning and Reasoning with Graph-Structured Representations*, 2019.
- [230] S. Verma and Z.-L. Zhang, “Hunt for the unique, stable, sparse and fast feature learning on graphs,” in *Advances in Neural Information Processing Systems*, 2017, pp. 88–98.
- [231] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal, “Graph2vec: Learning distributed representations of graphs,” *arXiv preprint arXiv:1707.05005*, 2017.
- [232] H. Chen and H. Koga, “Gl2vec: Graph embedding enriched by line graphs with edge features,” in *International Conference on Neural Information Processing*, Springer, 2019, pp. 3–14.
- [233] D. Noever and S. E. M. Noever, “Virus-mnist: A benchmark malware dataset,” *arXiv preprint arXiv:2103.00602*, 2021.

- [234] L. Nataraj, V. Yegneswaran, P. Porras, and J. Zhang, “A comparative assessment of malware classification using binary texture analysis and dynamic analysis,” in *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, 2011, pp. 21–30.
- [235] L. Chen, R. Sahita, J. Parikh, and M. Marino, “Stamina: Scalable deep learning approach for malware classification,” *Intel White Paper*,
- [236] J. Gennissen, L. Cavallaro, V. Moonsamy, and L. Batina, *Gamut: Sifting through images to detect android malware*, 2017.
- [237] K. Kancherla and S. Mukkamala, “Image visualization based malware detection,” in *2013 IEEE Symposium on Computational Intelligence in Cyber Security (CICS)*, IEEE, 2013, pp. 40–44.
- [238] S. Choi, S. Jang, Y. Kim, and J. Kim, “Malware detection using malware image and deep learning,” in *2017 International Conference on Information and Communication Technology Convergence (ICTC)*, IEEE, 2017, pp. 1193–1195.
- [239] J. Fu, J. Xue, Y. Wang, Z. Liu, and C. Shan, “Malware visualization for fine-grained classification,” *IEEE Access*, vol. 6, pp. 14 510–14 523, 2018.
- [240] K. S. Han, J. H. Lim, B. Kang, and E. G. Im, “Malware analysis using visualized images and entropy graphs,” *International Journal of Information Security*, vol. 14, no. 1, pp. 1–14, 2015.
- [241] J. Su, V. D. Vasconcellos, S. Prasad, S. Daniele, Y. Feng, and K. Sakurai, “Lightweight classification of iot malware based on image recognition,” in *2018 IEEE 42Nd annual computer software and applications conference (COMPSAC)*, IEEE, vol. 2, 2018, pp. 664–669.
- [242] T. C. of Economic Advisers, “The cost of malicious cyber activity to the u.s. economy,” 2018.
- [243] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, “Malware images: Visualization and automatic classification,” in *Proceedings of the 8th international symposium on visualization for cyber security*, 2011, pp. 1–7.
- [244] G. Conti, S. Bratus, A. Shubina, A. Lichtenberg, R. Ragsdale, R. Perez-Aleman, B. Sangster, and M. Supan, “A visual study of primitive binary fragment types,” *Black Hat USA*, 2010.
- [245] Y. Fang, Y. Gao, F. Jing, and L. Zhang, “Android malware familial classification based on dex file section features,” *IEEE Access*, vol. 8, pp. 10 614–10 627, 2020.

- [246] S. Lu, L. Ying, W. Lin, Y. Wang, M. Nie, K. Shen, L. Liu, and H. Duan, “New era of deeplearning-based malware intrusion detection: The malware detection and prediction based on deep learning,” *arXiv preprint arXiv:1907.08356*, 2019.
- [247] J.-S. Luo and D. C.-T. Lo, “Binary malware image classification using machine learning with local binary pattern,” in *2017 IEEE International Conference on Big Data (Big Data)*, IEEE, 2017, pp. 4664–4667.
- [248] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. K. Nicholas, “Malware detection by eating a whole exe,” in *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [249] N. McLaughlin, J. Martinez del Rincon, B. Kang, S. Yerima, P. Miller, S. Sezer, Y. Safaei, E. Trickel, Z. Zhao, A. Doupé, *et al.*, “Deep android malware detection,” in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, ACM, 2017, pp. 301–308.
- [250] F. Mercaldo and A. Santone, “Deep learning for image-based mobile malware detection,” *Journal of Computer Virology and Hacking Techniques*, 2020.
- [251] R. Burks, K. A. Islam, Y. Lu, and J. Li, “Data augmentation with generative models for improved malware detection: A comparative study,” in *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, IEEE, 2019, pp. 0660–0665.
- [252] A. Azab and M. Khasawneh, “Msic: Malware spectrogram image classification,” *IEEE Access*, vol. 8, pp. 102 007–102 021, 2020.
- [253] S. Yue, “Imbalanced malware images classification: A cnn based approach,” *arXiv:1708.08042*, 2017.
- [254] F. O. Catak, J. Ahmed, K. Sahinbas, and Z. H. Khand, “Data augmentation based malware detection using convolutional neural networks,” *arXiv preprint arXiv:2010.01862*, 2020.
- [255] Z. Ren, H. Wu, Q. Ning, I. Hussain, and B. Chen, “End-to-end malware detection for android iot devices using deep learning,” *Ad Hoc Networks*, 2020.
- [256] L. Chen, “Deep transfer learning for static malware classification,” *arXiv preprint arXiv:1812.07606*, 2018.
- [257] A. Jain, H. Gonzalez, and N. Stakhanova, “Enriching reverse engineering through visual exploration of android binaries,” in *Proceedings of the 5th Program Protection and Reverse Engineering Workshop*, 2015, pp. 1–9.

- [258] A. Kumar, K. P. Sagar, K. Kuppusamy, and G. Aghila, "Machine learning based malware classification for android applications using multimodal image representations," in *2016 10th international conference on intelligent systems and control (ISCO)*, IEEE, 2016, pp. 1–6.
- [259] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, "Microsoft malware classification challenge," *arXiv preprint arXiv:1802.10135*, 2018.
- [260] A. Nappa, M. Z. Rafique, and J. Caballero, "Driving in the cloud: An analysis of drive-by download operations and abuse reporting," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer, 2013, pp. 1–20.
- [261] S. Freitas, Y. Dong, J. Neil, and D. H. Chau, "A large-scale database for graph representation learning," *arXiv preprint arXiv:2011.07682*, 2020.
- [262] M. Kalash, M. Rochan, N. Mohammed, N. D. Bruce, Y. Wang, and F. Iqbal, "Malware classification with deep convolutional neural networks," in *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, IEEE, 2018, pp. 1–5.
- [263] Z. Cui, F. Xue, X. Cai, Y. Cao, G.-g. Wang, and J. Chen, "Detection of malicious code variants based on deep learning," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 7, pp. 3187–3196, 2018.
- [264] E. Rezende, G. Ruppert, T. Carvalho, A. Theophilo, F. Ramos, and P. de Geus, "Malicious software classification using vgg16 deep neural network's bottleneck features," in *Information Technology-New Generations*, Springer, 2018, pp. 51–59.
- [265] H. Yakura, S. Shinozaki, R. Nishimura, Y. Oyama, and J. Sakuma, "Malware analysis of imaged binary samples by convolutional neural network with attention mechanism," in *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, 2018, pp. 127–134.
- [266] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [267] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.
- [268] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

- [269] N. Bhodia, P. Prajapati, F. Di Troia, and M. Stamp, “Transfer learning for image-based malware classification,” *arXiv preprint arXiv:1903.11551*, 2019.
- [270] E. Rezende, G. Ruppert, T. Carvalho, F. Ramos, and P. De Geus, “Malicious software classification using transfer learning of resnet-50 deep neural network,” in *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, IEEE, 2017, pp. 1011–1014.
- [271] W. W. Lo, X. Yang, and Y. Wang, “An xception convolutional neural network for malware classification with transfer learning,” in *2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, IEEE, 2019, pp. 1–5.
- [272] X. Gao, C. Hu, C. Shan, B. Liu, Z. Niu, and H. Xie, “Malware classification for the cloud via semi-supervised transfer learning,” *Journal of Information Security and Applications*, vol. 55, p. 102 661, 2020.
- [273] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, “Focal loss for dense object detection,” in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.
- [274] H. Yakura, S. Shinozaki, R. Nishimura, Y. Oyama, and J. Sakuma, “Neural malware analysis with attention mechanism,” *Computers & Security*, vol. 87, p. 101 592, 2019.
- [275] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, “Grad-cam: Visual explanations from deep networks via gradient-based localization,” in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 618–626.
- [276] R. Zhao, W. Ouyang, H. Li, and X. Wang, “Saliency detection by multi-context deep learning,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1265–1274.
- [277] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” in *International Conference on Learning Representations*, 2014.
- [278] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, “Intriguing properties of neural networks,” in *ICLR*, 2014.
- [279] W. Brendel, J. Rauber, and M. Bethge, “Decision-based adversarial attacks: Reliable attacks against black-box machine learning models,” in *ICLR*, 2018.
- [280] S.-T. Chen, C. Cornelius, J. Martin, and D. H. Chau, “Shapeshifter: Robust physical adversarial attack on faster r-cnn object detector,” in *PKDD*, 2018.

- [281] A. Kurakin, I. J. Goodfellow, and S. Bengio, “Adversarial examples in the physical world,” *arXiv preprint arXiv:1607.02533*, 2016.
- [282] J. Su, D. V. Vargas, and K. Sakurai, “One pixel attack for fooling deep neural networks,” *arXiv preprint arXiv:1710.08864*, 2017.
- [283] A. Cabrera, F. Hohman, J. Lin, and D. H. Chau, “Interactive classification for deep learning interpretation,” *Demo, CVPR*, 2018.
- [284] F. Hohman, N. Hodas, and D. H. Chau, “Shapeshop: Towards understanding deep learning representations via interactive experimentation,” in *CHI*, 2017.
- [285] A. Ilyas, S. Santurkar, D. Tsipras, L. Engstrom, B. Tran, and A. Madry, “Adversarial examples are not bugs, they are features,” *arXiv preprint arXiv:1905.02175*, 2019.
- [286] D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, and A. Madry, “Robustness may be at odds with accuracy,” *arXiv preprint arXiv:1805.12152*, vol. 1050, p. 11, 2018.
- [287] X. Chen, R. Mottaghi, X. Liu, S. Fidler, R. Urtasun, and A. Yuille, “Detect what you can: Detecting and representing objects using holistic models and body parts,” in *CVPR*, 2014.
- [288] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, *The PASCAL Visual Object Classes Challenge 2010 (VOC2010) Results*.
- [289] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *CVPR*, 2009.
- [290] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft COCO: Common objects in context,” in *ECCV*, 2014.
- [291] M. Alzantot, Y. Sharma, S. Chakraborty, and M. B. Srivastava, “Genattack: Practical black-box attacks with gradient-free optimization,” *arXiv preprint*, 2018.
- [292] A. Ilyas, L. Engstrom, A. Athalye, and J. Lin, “Black-box adversarial attacks with limited queries and information,” in *ICML*, 2018.
- [293] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, “Practical black-box attacks against machine learning,” in *CCS*, 2017.
- [294] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, “Towards deep learning models resistant to adversarial attacks,” in *ICLR*, 2018.

- [295] Y. Dong, F. Liao, T. Pang, H. Su, J. Zhu, X. Hu, and J. Li, “Boosting adversarial attacks with momentum,” in *CVPR*, 2018, pp. 9185–9193.
- [296] A. Kurakin, I. J. Goodfellow, and S. Bengio, “Adversarial machine learning at scale,” in *International Conference on Learning Representations*, 2017.
- [297] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, “Ensemble adversarial training: Attacks and defenses,” in *ICLR*, 2018.
- [298] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, “Distillation as a defense to adversarial perturbations against deep neural networks,” in *IEEE Symposium on Security and Privacy*, IEEE, 2016, pp. 582–597.
- [299] N. Carlini and D. A. Wagner, “Defensive distillation is not robust to adversarial examples,” *arXiv preprint arXiv:1607.04311*, 2016.
- [300] N. Das, M. Shanbhogue, S.-T. Chen, F. Hohman, S. Li, L. Chen, M. E. Kounavis, and D. H. Chau, “Shield: Fast, practical defense and vaccination for deep learning using jpeg compression,” in *KDD*, 2018.
- [301] A. N. Bhagoji, D. Cullina, and P. Mittal, “Dimensionality reduction as a defense against evasion attacks on machine learning classifiers,” *arXiv preprint arXiv:1704.02654*, 2017.
- [302] R. Shin and D. Song, “Jpeg-resistant adversarial images,” *NIPS 2017 Workshop on Machine Learning and Computer Security*, 2017.
- [303] A. Athalye, N. Carlini, and D. Wagner, “Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples,” in *Proceedings of the 35th International Conference on Machine Learning*, 2018.
- [304] T. Gebhart and P. Schrater, “Adversary detection in neural networks via persistent homology,” *arXiv preprint arXiv:1711.10056*, 2017.
- [305] B. Liang, H. Li, M. Su, X. Li, W. Shi, and X. Wang, “Detecting adversarial examples in deep networks with adaptive noise reduction,” *arXiv preprint*, 2017.
- [306] J. Wang, J. Sun, P. Zhang, and X. Wang, “Detecting adversarial samples for deep neural networks through mutation testing,” *arXiv preprint*, 2018.
- [307] W. Xu, D. Evans, and Y. Qi, “Feature squeezing: Detecting adversarial examples in deep neural networks,” in *NDSS*, 2018.
- [308] F. Carrara, F. Falchi, R. Caldelli, G. Amato, R. Fumarola, and R. Becarelli, “Detecting adversarial example attacks to deep neural networks,” ser. CBMI, 2017.

- [309] X. Li and F. Li, “Adversarial examples detection in deep networks with convolutional filter statistics,” in *ICCV*, 2017, pp. 5775–5783.
- [310] D. Meng and H. Chen, “Magnet: A two-pronged defense against adversarial examples,” in *CSS*, ACM, 2017, pp. 135–147.
- [311] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask r-cnn,” in *ICCV*, 2017.
- [312] G. W. Ding, L. Wang, and X. Jin, “AdverTorch v0.1: An adversarial robustness toolbox based on pytorch,” *arXiv preprint arXiv:1902.07623*, 2019.
- [313] W. Abdulla, *Mask r-cnn for object detection and instance segmentation on keras and tensorflow*, https://github.com/matterport/Mask_RCNN, 2017.
- [314] A. Turner, D. Tsipras, and A. Madry, “Clean-label backdoor attacks,” 2018.
- [315] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *CVPR*, 2016, pp. 770–778.
- [316] A. Sors, S. Bonnet, S. Mirek, L. Vercueil, and J.-F. Payen, “A convolutional neural network for sleep stage scoring from raw single-channel eeg,” *Biomedical Signal Processing and Control*, vol. 42, pp. 107–114, 2018.
- [317] S. Biswal, J. Kulas, H. Sun, B. Goparaju, M. B. Westover, M. T. Bianchi, and J. Sun, “SLEEPNET: automated sleep staging system via deep learning,” *CoRR*, vol. abs/1707.08262, 2017. arXiv: 1707.08262.
- [318] A. Supratak, H. Dong, C. Wu, and Y. Guo, “Deepsleepnet: A model for automatic sleep stage scoring based on raw single-channel eeg,” *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 25, no. 11, pp. 1998–2008, 2017.
- [319] A. Sterr, J. K. Ebajemito, K. B. Mikkelsen, M. A. Bonmati-Carrion, N. Santhi, C. Della Monica, L. Grainger, G. Atzori, V. Revell, S. Debener, *et al.*, “Sleep eeg derived from behind-the-ear electrodes (ceeGRID) compared to standard polysomnography: A proof of concept study,” *Frontiers in human neuroscience*, vol. 12, p. 452, 2018.
- [320] A. Henriksen, M. H. Mikalsen, A. Z. Woldaregay, M. Muzny, G. Hartvigsen, L. A. Hopstock, and S. Grimsgaard, “Using fitness trackers and smartwatches to measure physical activity in research: Analysis of consumer wrist-worn wearables,” *Journal of medical Internet research*, vol. 20, no. 3, e110, 2018.
- [321] Z Beattie, A Pantelopoulos, A Ghoreyshi, Y Oyang, A Statan, and C Heneghan, “0068 ESTIMATION OF SLEEP STAGES USING CARDIAC AND ACCELEROM-

ETER DATA FROM a WRIST-WORN DEVICE,” *Sleep*, vol. 40, no. suppl_1, A26–A26, Apr. 2017.

- [322] K.-M. Chang and S.-H. Liu, “Gaussian noise filtering from ecg by wiener filter and ensemble empirical mode decomposition,” *Journal of Signal Processing Systems*, vol. 64, no. 2, pp. 249–264, 2011.
- [323] M. Blanco-Velasco, B. Weng, and K. E. Barner, “Ecg signal denoising and baseline wander correction based on the empirical mode decomposition,” *Computers in biology and medicine*, vol. 38, no. 1, pp. 1–13, 2008.
- [324] Y. Chen, M. Akutagawa, T. Emoto, and Y. Kinouchi, “The removal of emg in eeg by neural networks,” *Physiological measurement*, vol. 31, no. 12, p. 1567, 2010.
- [325] V. Bhateja, S. Urooj, R. Verma, and R. Mehrotra, “A novel approach for suppression of powerline interference and impulse noise in ecg signals,” in *IMPACT-2013*, IEEE, 2013, pp. 103–107.
- [326] S. Chambon, M. N. Galtier, P. J. Arnal, G. Wainrib, and A. Gramfort, “A deep learning architecture for temporal sleep stage classification using multivariate and multimodal time series,” *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 26, no. 4, pp. 758–769, 2018.
- [327] H. Phan, F. Andreotti, N. Cooray, O. Y. Chén, and M. De Vos, “Joint classification and prediction cnn framework for automatic sleep stage classification,” *IEEE Transactions on Biomedical Engineering*, vol. 66, no. 5, pp. 1285–1296, May 2019.
- [328] F. Andreotti, H. Phan, N. Cooray, C. Lo, M. T. M. Hu, and M. De Vos, “Multi-channel sleep stage classification and transfer learning using convolutional neural networks,” in *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, Jul. 2018, pp. 171–174.
- [329] M. Zhao, S. Yue, D. Katabi, T. S. Jaakkola, and M. T. Bianchi, “Learning sleep stages from radio signals: A conditional adversarial architecture,” *Proceedings of Machine Learning Research*, vol. 70, D. Precup and Y. W. Teh, Eds., pp. 4100–4109, 2017.
- [330] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” *arXiv preprint arXiv:1412.6572*, 2014.
- [331] N. Ford, J. Gilmer, N. Carlini, and D. Cubuk, “Adversarial examples are a natural consequence of test error in noise,” *CoRR*, vol. abs/1901.10513, 2019. arXiv:1901.10513.

- [332] D. Hendrycks and T. Dietterich, “Benchmarking neural network robustness to common corruptions and perturbations,” *arXiv preprint arXiv:1903.12261*, 2019.
- [333] S. Han, J. Pool, J. Tran, and W. Dally, “Learning both weights and connections for efficient neural network,” in *Advances in neural information processing systems*, 2015, pp. 1135–1143.
- [334] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, “Xnor-net: Imagenet classification using binary convolutional neural networks,” in *European Conference on Computer Vision*, Springer, 2016, pp. 525–542.
- [335] J. Xue, J. Li, and Y. Gong, “Restructuring of deep neural network acoustic models with singular value decomposition,” in *Interspeech*, 2013, pp. 2365–2369.
- [336] G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” *arXiv preprint arXiv:1503.02531*, 2015.
- [337] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, “Pruning filters for efficient convnets,” *arXiv preprint arXiv:1608.08710*, 2016.
- [338] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, “Learning structured sparsity in deep neural networks,” in *Advances in neural information processing systems*, 2016, pp. 2074–2082.
- [339] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, “Learning efficient convolutional networks through network slimming,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2736–2744.
- [340] R. Duggal, C. Xiao, R. Vuduc, and J. Sun, *Cup: Cluster pruning for compressing deep neural networks*, 2019. eprint: [arXiv:1911.08630](https://arxiv.org/abs/1911.08630).
- [341] Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo, Q. Wu, J. Huang, and J. Zhu, “Discrimination-aware channel pruning for deep neural networks,” in *Advances in Neural Information Processing Systems*, 2018, pp. 875–886.
- [342] Y. Guo, C. Zhang, C. Zhang, and Y. Chen, “Sparse dnns with improved adversarial robustness,” in *Advances in neural information processing systems*, 2018, pp. 242–251.
- [343] J. Lin, C. Gan, and S. Han, “Defensive quantization: When efficiency meets robustness,” *arXiv preprint arXiv:1904.08444*, 2019.
- [344] M. Cisse, P. Bojanowski, E. Grave, Y. Dauphin, and N. Usunier, “Parseval networks: Improving robustness to adversarial examples,” in *Proceedings of the 34th*

International Conference on Machine Learning-Volume 70, JMLR. org, 2017, pp. 854–863.

- [345] Y. Yoshida and T. Miyato, “Spectral norm regularization for improving the generalizability of deep learning,” *arXiv preprint arXiv:1705.10941*, 2017.
- [346] F. Farnia, J. M. Zhang, and D. Tse, “Generalizable adversarial training via spectral normalization,” *arXiv preprint arXiv:1811.07457*, 2018.
- [347] V. Lebedev and V. Lempitsky, “Fast convnets using group-wise brain damage,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2554–2564.
- [348] A. L. Goldberger, L. A. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley, “Physiobank, physiotoolkit, and physionet: Components of a new research resource for complex physiologic signals,” *Circulation*, vol. 101, no. 23, e215–e220, 2000.
- [349] S. F. Quan, B. V. Howard, C. Iber, J. P. Kiley, F. J. Nieto, G. T. O’Connor, D. M. Rapoport, S. Redline, J. Robbins, J. M. Samet, *et al.*, “The sleep heart health study: Design, rationale, and methods,” *Sleep*, vol. 20, no. 12, pp. 1077–1085, 1997.
- [350] R. B. Berry, R. Brooks, C. E. Gamaldo, S. M. Harding, C. L. Marcus, B. V. Vaughn, *et al.*, “The aasm manual for the scoring of sleep and associated events,” *Rules, Terminology and Technical Specifications*, Darien, Illinois, American Academy of Sleep Medicine, vol. 176, 2012.
- [351] S. Blanco, S. Kochen, O. Rosso, and P. Salgado, “Applying time-frequency analysis to seizure eeg activity,” *IEEE Engineering in medicine and biology magazine*, vol. 16, no. 1, pp. 64–71, 1997.