

UNIVERSITY OF PADUA

PH.D. SCHOOL IN BRAIN, MIND AND COMPUTER SCIENCE
Curriculum in Computer Science and Innovation for Societal Challenges
XXXV Cycle

Data-driven cybersecurity

Candidate:
Luca PAJOLA

Supervisor:
Prof. Mauro CONTI
University of Padua

Co-Supervisor:
Prof. Luciano Gamberini
University of Padua

Abstract

Due to the continuous growth in Internet data, cybersecurity practitioners have developed new defenses based on Machine Learning (ML). ML-based solutions offer numerous benefits, from learning patterns among large amounts of data to generalizing to unknown data. This dissertation covers three significant aspects derived from the interaction between machine learning and cybersecurity: (i) definition of novel Network Intrusion Detection Systems (NIDS), (ii) cybersecurity for web content monitoring, and (iii) Adversarial Machine Learning (AML).

The first part of the dissertation presents two NIDS themes: XeNIDS, aiming to study and design cross-networking NIDS, and DETONAR, a NIDS for low-powered IoT networks. The second part covers cybersecurity for web content monitoring. In particular, as users interact in forums and Online Social Networks (OSN), their activity might threaten others (e.g., hate speech). The dissertation covers two themes: helpful review prediction, aiming to forecast whether a review from forums (e.g., Amazon, Yelp) will be considered helpful, and PRaNA, a heuristic that leverages videos' Photo Response Non-Uniformity (PRNU) to spot real videos from their deepfake versions. The third - and last - part of the dissertation presents two evasion attacks: ZeW, an evasion attack on Natural Language Processing applications that leverages invisible UNICODE characters, and CAPA, which discusses real examples of threats created by OSN's users that undermined Automatic Content Moderators.

Contents

| | |
|---|------------|
| Abstract | iii |
| 1 Introduction | 1 |
| 1.1 Research Motivation and Contribution | 1 |
| 1.1.1 Publications | 5 |
| I Network Intrusion Detection Systems | 7 |
| 2 The Cross-evaluation of Machine Learning-based Network Intrusion Detection Systems | 9 |
| 2.1 Background | 11 |
| 2.1.1 Network Intrusion Detection and Machine Learning | 11 |
| 2.1.2 Motivation: Mixing Network Data | 12 |
| 2.1.3 Related Work | 14 |
| 2.2 Modelling the Cross-evaluation of ML-NIDS | 14 |
| 2.2.1 Proposed Model Design | 15 |
| 2.2.2 Benefits: additional Contexts | 15 |
| 2.2.3 Challenges and Risks | 19 |
| 2.3 Proposed Framework: XeNIDS | 20 |
| 2.3.1 Overview | 20 |
| 2.3.2 Standardize | 21 |
| 2.3.3 Isolate | 23 |
| 2.3.4 Contextualize | 24 |
| 2.3.5 Cross-evaluate | 25 |
| 2.4 Application | 26 |
| 2.4.1 Testbed | 26 |
| 2.4.2 Preprocessing | 28 |
| 2.4.3 Assessment | 29 |
| 2.5 Demonstration | 30 |
| 2.5.1 Baseline | 31 |
| 2.5.2 Generalization | 32 |
| 2.5.3 Extension | 33 |
| 2.5.4 Surrogation | 33 |
| 2.6 Discussion | 33 |
| 2.6.1 Preliminary Analysis | 34 |
| 2.6.2 Reliability: Uniform scenario | 34 |
| 2.6.3 Reliability: Heterogeneous scenario | 35 |
| 2.6.4 Limitations and Future Work | 35 |
| 2.7 Summary | 36 |
| 2.8 Appendix | 37 |
| 2.8.1 Contributors to NetFlows | 37 |
| 2.8.2 Guidelines for Standardize | 38 |

| | | |
|-----------|--|-----------|
| 2.8.3 | Symbol Table | 39 |
| 3 | DETONAR: Detection of Routing Attacks in RPL-based IoT | 41 |
| 3.1 | Background and Related Work | 43 |
| 3.1.1 | Routing Protocol for LLNs (RPL) | 43 |
| 3.1.2 | Attacks on RPL | 44 |
| 3.1.3 | State-of-the-art Intrusion Detection Systems | 45 |
| 3.2 | RADAR: Routing Attacks Dataset for RPL | 47 |
| 3.3 | Proposed RPL attacks detector: DETONAR | 48 |
| 3.3.1 | Overview | 49 |
| 3.3.2 | Traffic Collection | 49 |
| 3.3.3 | Features Extraction | 50 |
| 3.3.4 | Anomalies Detection | 51 |
| 3.3.5 | Attack Classification | 54 |
| 3.3.6 | Attacker Identification | 56 |
| 3.4 | Implementation and Performance evaluation | 58 |
| 3.4.1 | Implementation & Hyperparameters tuning | 58 |
| 3.4.2 | Results | 58 |
| 3.5 | Summary and Future work | 60 |
| II | The role of cybersecurity in web web platforms | 63 |
| 4 | A Novel Review Helpfulness Measure based on the User-Review-Item Paradigm | 65 |
| 4.1 | The Users-Reviews-Items (URI) Paradigm | 67 |
| 4.1.1 | The User-Review-Item (URI) Paradigm | 67 |
| 4.1.2 | User Graph | 69 |
| 4.1.3 | Review Graph | 70 |
| 4.1.4 | Item Graph | 73 |
| 4.1.5 | Multi Graphs | 73 |
| 4.1.6 | Summary | 75 |
| 4.2 | Prior Works | 75 |
| 4.3 | Experimental Settings | 81 |
| 4.3.1 | Feature Extraction Framework | 81 |
| 4.3.2 | Dataset | 83 |
| 4.4 | Temporal Analysis | 84 |
| 4.5 | Classification Results | 85 |
| 4.5.1 | Experimental Methodology | 85 |
| 4.5.2 | Classification Results | 86 |
| 4.5.3 | Cross-Domain Results | 86 |
| 4.5.4 | The impact of the ground-truth | 88 |
| 4.6 | Deep Neural Networks for Helpfulness Classification | 90 |
| 4.6.1 | Experimental Settings | 90 |
| 4.6.2 | Classification Results | 91 |
| 4.7 | Related Research Areas | 92 |
| 4.8 | Summary | 93 |

| | | |
|------------|---|------------|
| 5 | PRaNA: PRNU-based Technique to Tell Real and Deepfake Videos Apart | 95 |
| 5.0.1 | Organization | 96 |
| 5.1 | Background & Related Works | 97 |
| 5.1.1 | Photo Response Non-Uniformity | 97 |
| 5.1.2 | Deepfake Generation | 98 |
| 5.1.3 | Deepfake Detection | 98 |
| 5.2 | Datasets | 99 |
| 5.3 | PRaNA | 99 |
| 5.3.1 | Experimental Hypotheses | 100 |
| 5.3.2 | Autocorrelation Analysis | 100 |
| 5.3.3 | Considerations on recompression | 101 |
| 5.3.4 | Identification Algorithm | 103 |
| 5.3.5 | Frame sampling strategies | 103 |
| 5.4 | Results | 104 |
| 5.4.1 | Hyperparameter Search | 104 |
| 5.4.2 | Evaluation | 104 |
| 5.5 | Summary | 105 |
| | | |
| III | Adversarial Machine Learning | 107 |
| | | |
| 6 | Introduction to Adversarial Machine Learning | 109 |
| 6.1 | Preliminaries | 109 |
| 6.2 | Threats Landscape | 109 |
| 6.3 | Adversarial Transferability | 111 |
| | | |
| 7 | Fall of Giants: | |
| | How popular text-based MLaaS fall against a simple evasion attack | 113 |
| 7.1 | Preliminaries | 115 |
| 7.1.1 | Text Pipeline | 115 |
| 7.1.2 | Challenges of adversaries in Text-Domain | 117 |
| 7.2 | Zero-Width Attack | 118 |
| 7.2.1 | Motivations | 118 |
| 7.2.2 | Theoretical Perspective | 118 |
| 7.2.3 | Countermeasure | 120 |
| 7.3 | Case Study: Hate Speech Manipulation | 120 |
| 7.3.1 | Overview | 120 |
| 7.3.2 | Manipulation algorithm | 120 |
| 7.4 | Results on Controlled Environments | 121 |
| 7.4.1 | Experimental Settings | 121 |
| 7.4.2 | Results and Considerations | 122 |
| 7.5 | Results on MLaaS | 123 |
| 7.5.1 | Dataset & Evaluation on VaderSentiment | 124 |
| 7.5.2 | Hate Speech Detection | 124 |
| 7.5.3 | Insights Extractors | 126 |
| 7.5.4 | Sentiment Analyzers | 126 |
| 7.5.5 | Translators | 129 |
| 7.5.6 | Considerations | 129 |
| 7.6 | Related Work | 130 |
| 7.7 | Limitations | 131 |
| 7.8 | Summary | 132 |

| | | |
|----------|--|------------|
| 8 | Captcha Attack: Turning Captchas Against Humanity | 133 |
| 8.1 | Background & Related Works | 135 |
| 8.1.1 | Security of Machine Learning Applications | 135 |
| 8.1.2 | Moderators in OSN | 136 |
| 8.1.3 | CAPTCHA | 136 |
| 8.2 | Captcha Attack: A Taxonomy | 137 |
| 8.2.1 | Challenges from OSN's users: a Taxonomy | 137 |
| 8.2.2 | OCR-failure | 138 |
| 8.2.3 | Classifier-failures | 139 |
| 8.2.4 | Statistics from the wild | 139 |
| 8.3 | Attack Execution | 140 |
| 8.3.1 | Motivation | 140 |
| 8.3.2 | CC-CAPA Generation Procedure | 141 |
| 8.3.3 | CC-CAPA Dataset | 142 |
| 8.4 | Attack Results | 143 |
| 8.4.1 | Overview | 144 |
| 8.4.2 | Image Moderators | 144 |
| 8.4.3 | Cross-domain Moderators | 145 |
| 8.5 | CC-CAPA Detection Strategies | 146 |
| 8.5.1 | Overview | 146 |
| 8.5.2 | Supervised Approach: Classification | 147 |
| 8.5.3 | Unsupervised Approach: Outlier Detection | 149 |
| 8.5.4 | Toward Preventing CAPA | 151 |
| 8.5.5 | Comparison with State of the Art | 152 |
| 8.6 | Summary and Future Works | 152 |
| 8.7 | Appendix | 153 |
| 8.7.1 | CAPA Dataset | 153 |
| 8.7.2 | Captcha Schemes | 154 |
| 8.7.3 | Defense | 155 |
| 9 | Conclusion and Future Work | 157 |

List of Figures

| | | |
|-----|--|----|
| 2.1 | Typical NIDS scenario. The network of the organization can be composed of multiple subnetworks. The outgoing traffic passes through a border router which forwards such traffic to the internet, but also to a NIDS. The NIDS analyzes the traffic and, if necessary, raises some alerts. | 12 |
| 2.2 | Proposed cross-evaluation model, representing \mathbb{D} , \mathbb{N} and \mathbb{M} . Example: assume that \mathbb{D} has three datasets: D_1, D_2, D_3 (hence $n=3$). Assume that: D_1 has some benign samples but no malicious samples; D_2 has no benign samples, but malicious samples of a <i>botnet</i> attack, and also some malicious samples of a <i>portscan</i> attack; D_3 has some benign samples, alongside some malicious samples of the same <i>botnet</i> attack as D_2 , but also some malicious samples of a <i>DoS</i> attack. In this case, \mathbb{N} will contain three elements: N_1, N_2, N_3 , with N_2 being empty. On the other hand, there are three malicious classes ($\mu=3$) hence \mathbb{M} will have nine elements: M_2^1 and M_3^1 (the botnet attack shared by D_2 and D_3) as well as M_2^2 (the portscan attack in D_2) and M_3^3 (the DoS attack in D_3) will contain some samples; whereas the remaining will be empty (i.e., all those with M_1 , as well as M_2^3 and M_3^2). | 16 |
| 2.3 | Overview of XeNIDS. The input \mathbb{D} is a set of labelled NetFlow datasets of n distinct networks. The output results should be further analyzed. | 21 |
| 2.4 | First stage: Standardize. The initial NetFlows in \mathbb{D} are all standardized to derive a common feature set, and cleaned of any possible artifact that may lead to overfitting and impractical ML-NIDS. | 22 |
| 2.5 | Second stage: isolate. The standardized \mathbb{D} is used to extract the n individual benign N and malicious M sets. The latter are then split in μ sets on the basis of their attack class. Depending on the user-provided granularity, it is possible to aggregate sets of different attack classes into a bigger class, therefore changing the initial μ . The outputs are the full sets \mathbb{N} and \mathbb{M} | 23 |
| 2.6 | Third stage: contextualize. XeNIDS uses \mathbb{N} and \mathbb{M} to create multiple T and E , according to the context-related parameters $(\vec{o}, \vec{t}, \vec{e}, \vec{\tau}, \vec{\epsilon})$ and the splits $s(N)$ and $s(M)$. All the composed T and E are inserted in \mathbb{T} and \mathbb{E} . Example. Assume that the following user-provided parameters: $\vec{o}=(1), \vec{t}=(2), \vec{e}=(3), \vec{\tau}=(1), \vec{\epsilon}=(1); s(N)=(80:20), s(M)=(70:30)$. XeNIDS chooses N_1 , and puts 80% of N_1 in T and the remaining 20% in V . XeNIDS then selects M_2^1 and puts 70% of its samples in T ; then XeNIDS selects M_3^1 and puts 30% of its samples in V . Such T (and V) is then inserted in \mathbb{T} (and \mathbb{V}). The operation is repeated if the user provides additional lists as input. | 24 |

| | | |
|-----|---|----|
| 2.7 | Final stage: cross-evaluate. XeNIDS reproduces the user-specified context(s) and performs the cross-evaluation. Example: assume the following input context: $c((1), (1,2), (1,2), (1,1), (2,2))$ and a ML-NIDS that leverages ensembles of binary detectors. XeNIDS first extracts $T((1),(1,2),(1,1))$ from \mathbb{T} , which is split in two smaller sets, $T(1,1,1)$ and $T(1,2,1)$. Such sets are used to train two ML-models that will compose the ML-NIDS. The ML-NIDS can then be tested either against $E((1),(1,2),(2,2))$, or against its subsets $E(1,1,2)$ and $E(1,2,2)$, all of which obtained from \mathbb{E} . Any previously trained model (e.g., the one using $T(1,1,1)$) can be reused to assess different contexts. | 25 |
| 2.8 | Feature importances of the <i>Rbot</i> detectors (Heterogeneous CS). | 35 |
| 3.1 | DETONAR deploys sniffing devices to sense the network traffic and forward it to the centralized IDS server. IDS server is in charge of detecting possible anomalies, attacks, and compromised devices. . . . | 50 |
| 3.2 | DETONAR feature extraction overview. Feature vectors F_i^t are extracted for each node i at each time window W_i^t as the composition of the 11 representative features selected. Node behaviour B_i is built as the composition of feature vectors F_i^t | 52 |
| 3.3 | DETONAR extracts features from traffic received at a node to describe device's behavior. Feature series may differ significantly depending on considered feature and device. | 53 |
| 3.4 | ARIMA applied to the number of received DIO packets of a device in a legitimate and a sinkhole attack scenario. | 53 |
| 3.5 | DETONAR's decision flowchart. Each rule is based on considerations regarding RPL networks that help classify the final attack. Attacker identity depends on the attack classified. | 57 |
| 3.6 | ARIMA's false positives are influenced by DETONAR's time window size (i.e., ω), history size (i.e., λ) and test significance (i.e., α). | 58 |
| 4.1 | Example of an Amazon's product and its top review. | 66 |
| 4.2 | This figure illustrates the User-Review-Items (URI) Paradigm of a review platform. | 68 |
| 4.3 | Feature families popularity in prior works. | 69 |
| 4.4 | Temporal analyses on Toys and Game dataset. | 84 |
| 4.5 | Distribution of the Spearman Correlation between different feature categories and quarter number. For each distribution, we further report the median value. | 85 |
| 4.6 | Weighted F1-score (the higher, the better) among different datasets. . . | 87 |
| 4.7 | Average impact of features families (4.7a) and models (4.7b). | 87 |
| 4.8 | Cross dataset analysis of Random Forest-based models. On the left, the average F1-score at features level. In this graph, the models trained and tested on the same dataset are in blue, while cross-domain cases are shown in orange. On the right, the heat-map of models using 'ALL' features. The dataset name is reported with a compressed format. Starting from the left, Cell Phones and Accessories (CP), Digital Music (DM), Electronics (EL), Pet Supplies (PS), Toys and Games (TG), and Video Games (VG). | 88 |
| 4.9 | Long-tailed distribution of the 'number of votes' feature for the 'Toys and Games' dataset. | 89 |

| | | |
|------|---|-----|
| 4.10 | F1-score when varying of the ground-truth credibility. Starting from the left, Cell Phones and Accessories (CP), Digital Music (DM), Electronics (EL), Pet Supplies (PS), Toys and Games (TG), and Video Games (VG). | 90 |
| 5.1 | A frame related to a deepfake video of Barack Obama [300]. | 97 |
| 5.2 | Distribution of autocorrelation for 100 real videos and 100 fake videos from the DFDC dataset, computed using NCC. | 101 |
| 5.3 | Recompression analysis. | 102 |
| 5.4 | Accuracies at varying group sizes. | 105 |
| 7.1 | Zero-Width (ZeW) on a real-life scenario: Google Translate. The translated sentence means “I love you”. | 114 |
| 7.2 | Machine Learning pipeline in Natural Language Processing. | 115 |
| 7.3 | Negative sentiment densities of different corpus measured by Vader-Sentiment, where +1.0 is extremely negative and 0.0 is absence of negativity. The violin plot shows the distributions of the corpora’s negative scores; the blue line represents the median value of the distribution. The service is vulnerable if the distributions under attack are not equal to the distribution of <i>Real</i> | 124 |
| 7.4 | Toxicity score densities of different corpora measured by Google Perspective (left), and Microsoft Moderator (right), where +1.0 is high confidence of being classified as toxic. The violin plot shows the distributions of the corpora’s toxicity scores; the blue line represents the median value of the distribution. A service is vulnerable if the distributions under attack are not equal to the distribution of <i>Real</i> | 125 |
| 7.5 | The similarity distributions between <i>Real</i> vs. <i>Mask1</i> and <i>Real</i> vs. <i>Mask2</i> of Watson Tone Analyzer, where +1,0 is an exact match between two sets. The violin plot shows the distributions of the corpora’s Jaccard similarities; the blue line represents the median value of the distribution. The service is vulnerable if the distributions under attack are not close to one. | 127 |
| 7.6 | Watson Personal Insight detects three distinct personalities (<i>Real</i> , <i>Mask1</i> , and <i>Mask2</i>). The service is vulnerable if at least one of the five dimensions changes. | 127 |
| 7.7 | Effect of Zero-Width Space Attack on different sentiment extractor services. The violin plot shows the distributions of the corpora’s negative scores; the blue line represents the median value of the distribution. A service is vulnerable if the distributions under attack are not equal to the distribution of <i>Real</i> | 128 |
| 7.8 | Effect of Zero-Width Space Attack on different translator services. The violin plot shows the distributions of the corpora’s BLEU scores; the blue line represents the median value of the distribution. A service is vulnerable if the distributions under attack are not close to one. | 130 |
| 8.1 | Instagram alert of sensitive content. | 134 |
| 8.2 | Example of meme with different obfuscations (e.g., typos, letters-shaped objects, hard background). | 134 |
| 8.3 | Overview of a content moderator in the text and image domains. | 137 |

| | | |
|------|---|-----|
| 8.4 | Obfuscation techniques we identified in online social networks. Blue boxes represents the ACM component that might fails. Green boxes represent different obfuscation techniques. We censored explicit or harmful contents with the red symbol. | 138 |
| 8.5 | Overview of CC-CAPA execution pipeline. | 142 |
| 8.6 | Captchas' styles used in the experiments. | 143 |
| 8.7 | Cross-domain evaluation. We report the Attack Success Rate (ASR) (the higher, the better). | 146 |
| 8.8 | T-SNE 2D visualization of 2000 samples benign (Pinterest) and 2000 captchas (C11 and CAPA). | 149 |
| 8.9 | F1-score of different Outlier Detection at the varying of the OSN. | 149 |
| 8.10 | Example of false positive found among the outliers that should be moderated. | 151 |
| 8.11 | User-study performance distribution. We report accuracy (the higher, the better), and CER (the lower, the better). | 154 |
| 8.12 | Cross-domain evaluation. On the left, the Attack Success Rate (ASR). On the right, the average Normalized Levenshtein Distance (NLD). For both measures, the higher, the more successful the attack. | 156 |
| 8.13 | Accuracy of different Outlier Detection on known captcha styles at the varying of the OSN. | 156 |
| 8.14 | Accuracy of different Outlier Detection on unknown captcha styles at the varying of the OSN. | 156 |

List of Tables

| | | |
|-----|---|----|
| 2.1 | The 10 types of contexts enabled by the proposed cross-evaluation model. The table represents the relationship among the unique elements of \mathbb{N} and \mathbb{M} included in the training and evaluation sets, T and E (cf. Expression 2.2), which denote a context (cf. Expression 2.3). The origin of benign samples is always the same in T and E , hence \bar{o} is shared. Contexts of type c_1, c_2 are those considered by most prior works; c_4 has been considered in [234]. A gray background denotes the types of context assessed in our demonstration. Example: consider the setting described in the example of Figure 2.2, where $n=3$ and $\mu=3$. Suppose a context denoted by: $\bar{o}=(3), \bar{i}=(2,3), \bar{e}=(3), \bar{\tau}=(1,1), \bar{\epsilon}=(1)$. Such context implies that T contains benign samples from N_3 , and malicious samples from M_2^1 and M_3^1 ; whereas V contains benign samples from N_3 but malicious samples from M_3^1 . The resulting context will be of type c_5 , because $\bar{o}=(3), \bar{i}=(2,3), \bar{e}=(3), \bar{\tau}=(1), \bar{\epsilon}=(1)$. Hence, $(\bar{o} = \bar{e}) \neq \bar{i}$ and $\bar{\tau} = \bar{\epsilon}$ | 18 |
| 2.2 | Statistics of the analyzed NID datasets. | 27 |
| 2.3 | Distribution of attacks in each dataset. In our implementation of XeNIDS, we <i>always</i> use the specific attack classes, and perform no merging. | 28 |
| 2.4 | Feature set of our XeNIDS implementation in both scenarios. | 29 |
| 2.5 | Baseline- c_1 . The table shows the F1-score of the ML-NIDS for each origin network. The performance matches the state-of-the-art. The FPR is less than 0.001 for all networks aside from $UF-BotIoT$ (FPR = 0.11). | 31 |
| 2.6 | Generalization- c_4 . Each baseline ML-NIDS of c_1 is tested against the attacks of all other networks. Most attacks are not detected, and the F1-score degrades. The FPR is the same as in c_1 because the benign samples are always the same and the training set is not modified. | 32 |
| 2.7 | Extension- c_7 . By augmenting the training set of the ML-NIDS with the malicious samples, the F1-score improves w.r.t. c_4 . The average FPR is lower than 0.001 for all networks aside from $UF-BotIoT$ (FPR = 0.01). | 33 |
| 2.8 | Surrogation- c_7 . We exclude all malicious samples (and detectors) from the each ‘origin’ network. The extremely high performance must be investigated. The average FPR is less than 0.001 and 0.0001 for all networks of <i>uniform</i> and <i>heterogeneous</i> scenarios, respectively. | 33 |
| 2.9 | Table of relevant notation used in this paper. | 40 |
| 3.1 | Main features of attacks against RPL. These features help to understand attacks behaviour and to classify them. | 45 |
| 3.2 | State-of-the-art IDSs and corresponding detected attacks. We can see that several attacks are not covered. | 46 |
| 3.3 | RADaR’s simulated scenarios and the corresponding average amount of packets collected. For each packet specific features like source, destination, rank, etc. are collected. | 48 |

| | | |
|-----|---|-----|
| 3.4 | Selected features of our IDS. It is possible to notice that each feature helps detect one or more attacks. | 51 |
| 3.5 | False positives for Anomaly Detection (AD) and AD + Attack Classification (AC) of DETONAR in five legitimate simulations. | 59 |
| 3.6 | DETONAR can detect most attacks with satisfactory accuracy both in terms of attack detection (DET) and attacker identification (ID). In wormhole, ID contains the number of attackers identified correctly out of the two existing. | 60 |
| 4.1 | Description of features available with our framework. | 81 |
| 4.2 | Dataset statistics. We show dimension and percentage of helpful reviews for each set. | 84 |
| 4.3 | F1-scores with varying the ground-truth credibility on the following Amazon categories: Cell Phones and Accessories (CP), Digital Music (DM), Electronics (EL), Pet Supplies (PS), Toys and Games (TG), and Video Games (VG). This table contains the performance of the following classifiers: Adaboost (AB), Logistic Regression (LR), Random Forest (RF), Graph Neural Network (GNN), DistilBert (DB), and MultiLayer Perceptron (MLP). Highlighted, the best F1 scores. | 92 |
| 5.1 | Accuracies for real video identification at various parameters on the DFDC dataset. In Bold the best accuracy. N/A for settings with not enough frames to run the task. | 104 |
| 7.1 | Overview of models' performance. The accuracy score (ACC) measure the quality of the model on the three splits. The attack success percentage (ASP) measures the misclassification percentage of a given classifier; in bold the results of models resistant to ZeW. | 123 |
| 8.1 | Percentage of obfuscation techniques observed in different Instagram sources. | 140 |
| 8.2 | Datasets' statistics. | 148 |
| 8.3 | Avg retrieval results of 11 captcha schemes in different OSNs. | 148 |
| 8.4 | List of transformations for textual captchas variants. | 153 |
| 8.5 | Captcha schemes used in our experiment coming from [320]. | 155 |
| 8.6 | Percentage of \mathbb{D}_{tox}^m captchas detected by models trained on data coming from different OSNs. | 156 |

Chapter 1

Introduction

Cybersecurity is the discipline of protecting digital systems from digital attacks aiming to compromise them. Generally speaking, a cybersecurity incident involves an intruder using a tool to exploit a vulnerability on an objective, resulting in an outcome that suits the attacker's needs. Continuous investment in cybersecurity is needed as cyber-attacks constantly evolve, threatening many organizations worldwide. Mirai botnet is only one example of a successful attack that disrupted the Internet [10]. Or similarly, ransomware campaigns undermine worldwide private and public organizations. The number of ransomware attacks are severe: in 2020 estimated at 304 million, while in 2021, 623 million [270]. Cyber-attacks further evolved from more traditional system intrusions to threats to users, such as Twitter trends manipulations [328] and the spread of fake news in online social networks [326].

Cybersecurity tools thus aim to inspect an enormous amount of data generated on the Internet. Such flows are the results of the communication of networks or users. As humans cannot keep up analyzing the constantly increasing amount of data, traditional approaches based on rules are thus gradually replaced by automated approaches such as Machine Learning (ML) [280, 68, 243]. ML-based strategies are perfect for working with big data and deriving knowledge from it. For example, signature-based and heuristic-based detection approaches are efficient in detecting known malware but fail to detect unknown ones. This limitation can be mitigated with ML-based solutions [17].

There has been a proliferation of successful data-driven approaches in cybersecurity literature. For example, Drebin [16] is an Android Malware detection that extracts statistics from run-time applications and uses such information to predict whether an application is malicious. Drebin reaches noticeable performance, with malware detection close to 93% for known, maintaining a low false alarm (1%). Furthermore, Drebin requires only a few samples of a new malware family to produce reliable detection results. Alternatively, complex and unsolved threats are currently investigated with ML techniques: popular examples are the fake-news and hate-speech detection [265, 192].

1.1 Research Motivation and Contribution

This thesis investigates the interplay between cybersecurity and data-driven approaches, focusing on three major aspects.

1. *Network Intrusion Detection Systems*: solutions aiming to spot malicious activities from the network traffic. Chapter 2 presents XeNIDS, a study aiming to investigate benefits derived from the cross-evaluation of NIDS models among different datasets. Chapter 3 presents DETONAR, a NIDS for IoT networks using RPL protocol.

2. *Cybersecurity for web content monitoring*: solutions aiming to analyze content generated by users' interactions in web applications. Chapter 4 presents a study of the reviews in online forums (e.g., Amazon, TripAdvisor), aiming to understand in advance whether a review is helpful. Chapter 5 presents PRaNA, a forensic tool to analyze deepfake videos.
3. *Adversarial Machine Learning*: the study of ML-based solutions security. Chapter 7 presents ZeW, an evasion attack to Natural-Language-Processing (NLP) applications. Chapter 8 presents CAPA, an evasion attack to Automatic Content Moderators.

In this dissertation, some passages have been quoted verbatim, and some figures have been reused from the works [15, 4, 220, 66], all coauthored by the author of the thesis.

The Cross-evaluation of Machine Learning-based Network Intrusion Detection Systems

Enhancing Network Intrusion Detection Systems (NIDS) with supervised Machine Learning (ML) is tough. ML-NIDS must be trained and evaluated, operations requiring data where benign and malicious samples are clearly labelled. Such labels demand costly expert knowledge, resulting in a lack of real deployments, as well as on papers always relying on the same outdated data. The situation improved recently, as some efforts disclosed their labelled datasets. However, most past works used such datasets just as a 'yet another' testbed, overlooking the added potential provided by such availability.

Contributions In contrast, in Chapter 2 we promote using such existing labelled data to *cross-evaluate* ML-NIDS. Such approach received only limited attention and, due to its complexity, requires a dedicated treatment. We hence propose the first cross-evaluation model. Our model highlights the broader range of realistic use-cases that can be assessed via cross-evaluations, allowing the discovery of still unknown qualities of state-of-the-art ML-NIDS. For instance, their detection surface can be extended—at no additional labelling cost. However, conducting such cross-evaluations is challenging. Hence, we propose the first framework, XeNIDS, for reliable cross-evaluations based on Network Flows. By using XeNIDS on six well-known datasets, we demonstrate the concealed potential, but also the risks, of cross-evaluations of ML-NIDS.

DETONAR: Detection of Routing Attacks in RPL-based IoT

The Internet of Things (IoT) is a reality that changes several aspects of our daily life, from smart home monitoring to the management of critical infrastructure. The "Routing Protocol for low power and Lossy networks" (RPL) is the only de-facto standardized routing protocol in IoT networks and is thus deployed in environmental monitoring, healthcare, smart building, and many other IoT applications. In literature, we can find several attacks aiming to affect and disrupt RPL-based networks. Therefore, it is fundamental to develop security mechanisms that detect and mitigate any potential attack in RPL-based networks. Current state-of-the-art security solutions deal with very few attacks while introducing heavy mechanisms at the expense of IoT devices and the overall network performance.

Contribution In Chapter 3, we aim to develop an Intrusion Detection System (IDS) capable of dealing with multiple attacks while avoiding any RPL overhead. The proposed system is called DETONAR - DETector of rOutiNg Attacks in Rpl - and it relies on a packet sniffing approach. DETONAR uses a combination of signature and anomaly-based rules to identify any malicious behavior in the traffic (e.g., application and DIO packets). To the best of our knowledge, there are no exhaustive datasets containing RPL traffic for a vast range of attacks. To overcome this issue and evaluate our IDS, we propose RADAR - Routing Attacks DATaset for Rpl: the dataset contains five simulations for each of the 14 considered attacks in 16 static-nodes networks. DETONAR's attack detection exceeds 80% for 10 attacks out of 14, while maintaining false positives close to zero.

A Novel Review Helpfulness Measure based on the User-Review-Item Paradigm

Review platforms are viral online services where users share and read opinions about products (e.g., a smartphone) or experiences (e.g., a meal at a restaurant). Other users may be influenced by such opinions when making future choices. The usability of review platforms is currently limited by the massive number of opinions on many products. Therefore, showing only the most *helpful* reviews for each product is in the best interests of both users and the platform (e.g., Amazon). The current state of the art is far from accurately predicting how helpful a review is. First, most existing work lacks compelling comparisons as many studies are conducted on datasets that are not publicly available. As a consequence, new studies are not always built on top of prior baselines. Second, most research focuses only on features derived from the review text, ignoring other fundamental aspects of the review platforms (e.g., the other reviews of a product, the order in which they were submitted).

Contribution In Chapter 4, we first carefully review the most relevant works in the area published during the last 20 years. We then propose the User-Review-Item (URI) paradigm, a novel abstraction for modeling the problem that moves the focus of the feature engineering from the review to the platform level. We empirically validate the URI on a dataset of products from six Amazon categories with 270 trained models: on average, classifiers gain +5% in F1-score when considering the whole review platform context. In our experiments, we further emphasize some problems with the helpfulness prediction task: (1) the users' writing style changes over time (i.e., concept drift), (2) past models do not generalize well across different review categories, and (3) past methods to generate the ground-truth produced unreliable helpfulness scores, affecting the model evaluation phase.

PRaNA: PRNU-based Technique to Tell Real and Deepfake Videos Apart

Videos are a powerful source of communication adopted in several contexts and used for both benign and malicious purposes (e.g., education vs. reputation damage). Nowadays, realistic video manipulation strategies like deepfake generators constitute a severe threat to our society in term of misinformation. While the wide range of the current research focuses on deepfake detection as a binary task, the identification of a real video among a pool of deepfakes sharing the same origin is not widely investigated. While the pool task might be more rare in real-life compared to the binary one, outcomes that derives from these analyses might let us better understand deepfake behaviours, benefiting binary deep fake detection as well.

Contribution In Chapter 5, we address the less investigated scenario by investigating the role of Photo Response Non-Uniformity (PRNU) in deepfake detection. Our analysis, in agreement with prior studies, shows that PRNU can be a valuable source to identify deepfake videos. In particular, we found that unique PRNU characteristics exist to distinguish real videos from their deepfake versions: real video autocorrelations tend to be lower compared to their deepfakes versions. Motivated by this, we propose PRaNA, a training-free strategy that leverages PRNU autocorrelation. Our results on three well-known datasets confirm our algorithm’s robustness and transferability, with accuracy up to 66% when considering one real video in a pool of four deepfakes using the real video as a source, and up to 80% when only one deepfake is considered. Our work aims to open different strategies to counter deepfake diffusion.

Fall of Giants: How popular text-based MLaaS fall against a simple evasion attack

The increased demand for machine learning applications made companies offer Machine-Learning-as-a-Service (MLaaS). In MLaaS (a market estimated 8000M USD by 2025), users pay for well-performing ML models without dealing with the complicated training procedure. Among MLaaS, text-based applications are the most popular ones (e.g., language translators). Given this popularity, MLaaS must provide resiliency to adversarial manipulations. For example, a wrong translation might lead to a misunderstanding between two parties. In the text domain, state-of-the-art attacks mainly focus on strategies that leverage ML models’ weaknesses. Unfortunately, not much attention has been given to the other pipeline’ stages, such as the indexing stage (i.e., when a sentence is converted from a textual to a numerical representation) that, if manipulated, can significantly affect the final performance of the application.

Contribution In Chapter 7, we propose a novel text evasion technique called “Zero-Width attack” (ZeW) that leverages the injection of human non-readable characters, affecting indexing stage mechanisms. We demonstrate that our simple yet effective attack deceives MLaaS of “giants” such as Amazon, Google, IBM, and Microsoft. Our case study, based on the manipulation of hateful tweets, shows that out of 12 analyzed services, only one is resistant to our injection strategy. We finally introduce and test a simple *input validation* defense that can prevent our proposed attack.

Captcha Attack: Turning Captchas Against Humanity

The field of Adversarial Machine Learning studies the security related to Machine Learning (ML) systems. There is little information available about attacks happening in the real world, whereas most of the literature presents experiments in artificial settings. Such a gap must be filled urgently. In this work, we focus on the security of Automatic Content Moderator (ACM), i.e., deployed ML-powered technologies aiming to prevent the spreading of inappropriate content (e.g., hate speech, nudity images) in forums and social networks.

Contribution In Chapter 8, we analyzed 4600 potentially toxic Instagram posts, and we discovered that 44% of them adopt obfuscations that might undermine

ACM. As these posts are reminiscent of captchas (i.e., not understandable by automated mechanisms), we coin this threat as Captcha Attack (CAPA). Our contributions start by proposing a CAPA taxonomy to better understand how ACM is vulnerable to obfuscation attacks. We then focus on the broad sub-category of CAPA using textual Captcha Challenges, namely CC-CAPA, and we empirically demonstrate that it evades real-world ACM (i.e., Amazon, Google, Microsoft) with 100% accuracy. Our investigation revealed that ACM failures are caused by the OCR text extraction phase. The training of OCRs to withstand such obfuscation is therefore crucial, but huge amounts of data are required. Thus, we investigate methods to identify CC-CAPA samples from large sets of data (originated by three OSN – Pinterest, Twitter, Yahoo-Flickr), and we empirically demonstrate that supervised techniques identify target styles of samples almost perfectly. Unsupervised solutions, on the other hand, represent a solid methodology for inspecting uncommon data to detect new obfuscation techniques.

1.1.1 Publications

This section summarizes manuscripts produced during my Ph.D. period and published or currently submitted in peer-reviewed journal and conferences. All manuscript are listed in a chronological order of acceptance and submission.

Journal Publication

1. Agiollo, A., Conti, M., Kaliyar, P., Lin, T. N., & Pajola, L. (2021). DETONAR: Detection of routing attacks in RPL-based IoT. *IEEE Transactions on Network and Service Management*, 18(2), 1178-1190. (JCR IF 2019: 5.333)
2. Apruzzese, G., Pajola, L., & Conti, M. (2022). The Cross-evaluation of Machine Learning-based Network Intrusion Detection Systems. *IEEE Transactions on Network and Service Management*. (JCR IF 2020: 3.878)
3. Pajola, L., Chen, H., Conti, M., & Subrahmanian, V.S. (2022). A Novel Review Helpfulness Measure based on the User-Review-Item Paradigm. *ACM Transactions on the Web*. (JCR IF 2022: 4.235) Submitted

Conference Publication

- Cardaioli, M., Ceconello, S., Conti, M., Pajola, L., & Turrin, F. (2020, September). Fake News Spreaders Profiling Through Behavioural Analysis. In *CLEF (Working Notes)*.
- Mari, D., Piazzetta, S. G., Bordin, S., Pajola, L., Verde, S., Milani, S., & Conti, M. (2021, June). Looking Through Walls: Inferring Scenes from Video-Surveillance Encrypted Traffic. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 2595-2599). IEEE. (CORE:B, LiveSHINE:A+, MA:A+, Italian GGS 2/A)
- Pajola, L., & Conti, M. (2021, September). Fall of Giants: How popular text-based MLaaS fall against a simple evasion attack. In *2021 IEEE European Symposium on Security and Privacy (EuroS&P)* (pp. 198-211). IEEE. (MA:C)
- Amerini, I., Conti, M., & Pajola, L. PRaNA: PRNU-based Technique to Tell Real and Deepfake Videos Apart. In *Proceedings of the IEEE World Congress on*

Computational Intelligence (WCCI 2022), in press, Padua, Italy, July 18-23, 2022. **(LiveSHINE:C, MA:B)**

- Conti, M., Pajola, L., & Tricomi, P. P. (2022). *Captcha Attack: Turning Captchas Against Humanity*. *IEEE Conference on Secure and Trustworthy Machine Learning*. Submitted
- Nadeem, A., Vos, D., Cao, C., Pajola, L., Dieck, S., Baumgartner, R., & Verwer, S. (2022). *SoK: Explainable Machine Learning for Computer Security Applications*. *IEEE Symposium on Security and Privacy*. **(CORE:A++, LiveSHINE:A++, MA:A++, Italian GGS: 1/A++)** Submitted
- Tricomi, P. P., Nenna, F., Pajola, L., Conti, M., & Gamberini, L. *You Can't Hide Behind Your Headset: User Profiling in Augmented and Virtual Reality*. *ACM Conference on Human Factors in Computing Systems (CHI)*. **(CORE:A++, LiveSHINE:A++, MA:A++, Italian GGS: 1/A++)** Submitted

Part I

Network Intrusion Detection Systems

Chapter 2

The Cross-evaluation of Machine Learning-based Network Intrusion Detection Systems

Machine Learning (ML) is advancing at a rapid pace (e.g., [266, 311]), and the cybersecurity domain is also looking at ML with great interest [37]. ML methods can automatically learn to make decisions by using existing data, representing a valuable asset to monitor the increasingly mutating IT environments.

Although ML is already deployed to counter some threats (e.g., malware or phishing [92, 85, 176]), ML methods are still at an early stage for Network Intrusion Detection (NID). In particular, some Network Intrusion Detection Systems (NIDS) integrate commercial products that use *unsupervised* ML (e.g., [72, 167]). Such solutions can be useful to perform correlation analyses or to ‘detect anomalies’, which are ancillary to true intrusion detection tasks (an anomaly is not necessarily an intrusion). The full potential of ML can be appreciated only via *supervised* methods, which assume the existence of *labels* that associate each sample to its ground truth [269]. Specifically in NID, by creating a training dataset where the samples are distinguished between *benign* and *malicious*, it is possible to develop a fully autonomous Machine Learning-based Network Intrusion Detection System (ML-NIDS).

Deployment of ML-NIDS involves two stages: the system must first be developed (i.e., *trained*), and it must then be *evaluated*, because any security system that has not been tested is dangerous [28]. Both of these stages require *large* amounts of *labelled* data, which can only be collected via the supervision of a human that associates (and verifies) each sample to its ground truth [12]. While such verifications are simple in some applications (e.g., any layman can distinguish a legitimate from a phishing website), the inspection of network data requires expert knowledge—which is expensive [203]. To aggravate the problem, a network can be targeted by many attacks, *each* of which must be labelled to assess the detection capabilities of a ML-NIDS. As a result, the inevitable and costly necessity of comprehensive labelled datasets (usually numbering millions of samples [247]) discourages deployment of ML-NIDS. We note that this problem also extends to *research*. For more than a decade, the only publicly available dataset for ML-NIDS was the KDD99, leading to a plethora of works always trained and evaluated on such dataset—usually with perfect performance (e.g., [206]).

To address the lack of labelled data, recent researches on ML-NIDS openly released their datasets (e.g., [247, 262, 101]), an effort appreciated by related literature (e.g., [206, 11, 135]). However, most prior works used such datasets as an additional testbed for their proposals. As a result, such works only confirmed what was already known: that by training a ML-NIDS on a (large) dataset, such ML-NIDS will detect

the attacks contained in such dataset. This is because the primary objective was to ‘outperform’ the state-of-the-art, resulting in incremental contributions that do not foster realistic deployments. In this paper, we aim to broaden such limited scope.

Inspired by a recent paper by Pontes et al. [234], we observe that the current availability of labelled datasets could be better exploited by ML-NIDS researches. Specifically, we endorse the idea of *cross-evaluating* ML-NIDS by using malicious samples captured in different network datasets.¹ By performing such cross-evaluations, it is possible to gauge additional properties of ML-NIDS, allowing a better understanding of the state-of-the-art at no extra labelling cost.

To the best of our knowledge, this is the first effort that focuses on the opportunity provided by cross-evaluations of ML-NIDS. As such, our primary goal is the definition of a data-agnostic *model* that allows to represent such cross-evaluations. Indeed, using samples from different networks is not straightforward: as stated by Sommer and Paxson [269], each network has “immense variability”, suggesting that cross-evaluations have intrinsic risks that must be known to avoid deployments of unreliable ML-NIDS. Our model acknowledges such risks, but also highlights the *benefits* that can be brought by cross-evaluations of ML-NIDS. Such benefits come in the form of additional types of ‘contexts’ that can be reproduced in research environments, each representing a distinct realistic use-case. Specifically, our model highlights the limited scope of the state-of-the-art, whose fixed evaluation methodology can only cover 2 contexts, whereas cross-evaluations can span over up to 10 different contexts. Such broad range evidences the concealed potential of the core idea at the base of our paper.

As stated by Biggio et al. [28], ML systems for cybersecurity must be assessed *in advance*. Therefore, *proactive* cross-evaluations must take into account all the pitfalls highlighted by our model. To further promote our proposal, we develop the first framework for cross-evaluations of ML-NIDS, XeNIDS. Our framework aims to overcome the intrinsic challenges of cross-evaluations, while allowing the reproduction of all contexts enabled by our model. Specifically, XeNIDS focuses on NetFlow data, which is popular in the ML-NIDS community (e.g., [255, 234, 247]). However, using NetFlows from different environments is tough: such data can be generated in many ways, resulting in heterogeneous formats that may lead to unreliable ML-NIDS. We address this issue via an *original* interpretation of NetFlows w.r.t. ML. Using this interpretation, we provide the guidelines that can increase the reliability of the results provided by XeNIDS.

As an instructive *demonstration*, we use XeNIDS to perform a large cross-evaluation of ML-NIDS spanning over 6 well-known and recent datasets. We aim to reproduce realistic use cases, which can be assessed via three different context types enabled by our model. Specifically, we first consider the ‘baseline’ context commonly adopted by prior work, and show that XeNIDS yields the same performance as the state-of-the-art. Then, we assess the context where a ML-NIDS is *tested* on malicious samples originating from different networks; such use-case was also investigated in [234], and XeNIDS matches their performance. Finally, we assess the context where the ML-NIDS is *trained and tested* on malicious samples from different networks, showing a dramatic performance increase. As a final contribution of this paper, we provide an in-depth *analysis* of these results, where we investigate their reliability for practical deployments.

¹We stress that our term ‘cross-evaluation’ denotes a different concept than the term ‘cross-validation’ commonly used in ML researches [256].

Contribution This is the first paper that addresses the problem of cross-evaluations of ML-NIDS. As such, the specific contributions are as follows.

- We present the first data-agnostic model that conceptualizes the problem of cross-evaluation of ML-NIDS.
- We use our model to showcase the benefits and challenges of such cross-evaluations.
- We propose XeNIDS, the first framework for reliable cross-evaluations focused on NetFlow data.
- We demonstrate all of the above by cross-evaluating ML-NIDS over 6 distinct well-known datasets, and analyzing the results' reliability.

Organization The chapter is organized as follows. We motivate our paper in Section 2.1. We define our cross-evaluation model in Section 2.2. We describe our XeNIDS framework in Section 2.3. We explain the application of XeNIDS in Section 2.4. We present our demonstration in Section 2.5. We discuss the results in Section 2.6. We conclude our paper in Section 2.7.

2.1 Background

This work lies at the intersection of Machine Learning and Network Intrusion Detection. We first provide some preliminary information on these two areas (Section 2.1.1). Then, we explain the motivation (Section 2.1.2) of our paper. Finally, we compare this effort with related work (Section 2.1.3).

2.1.1 Network Intrusion Detection and Machine Learning

The so-called security lifecycle spans over three activities: prevention, detection, reaction [307]. However, the prevention of any cyber-attack is an impossible task, while the reaction phase assumes that most of the damage has already taken place. For this reason, proposals focusing on the detection step have received much more attention, as timely and accurate identifications of cyber threats can significantly mitigate the effects of an offensive campaign [231].

In the specific domain of network security (which is of interest to this paper), the detection of such malicious events is devoted to Network Intrusion Detection Systems (NIDS). We provide a schematic representation of the typical NIDS deployment in Figure 2.1, where a NIDS inspects the traffic generated by the monitored network (and all of its subnetworks). A NIDS can leverage two distinct detection paradigms, which are based either on *fixed rules* or on *data-driven* methods [39]. The former requires human operators that write specific rules (or signatures) that denote a specific threat, and exhibit high performance against *known* and *static* threats whose behavior is captured by the hardcoded rules. On the other hand, the latter leverage automatic data analyses and can detect even *unknown* and *mutating* threats if they present similarities with previously known samples—potentially at the expense of higher false-positive rates.

The increased growth of data alongside improvements in collaborative computing resulted in a huge interest in data-driven NIDS, specifically employing machine learning methods [12, 37]. Such methods involve a *training* phase where the ML

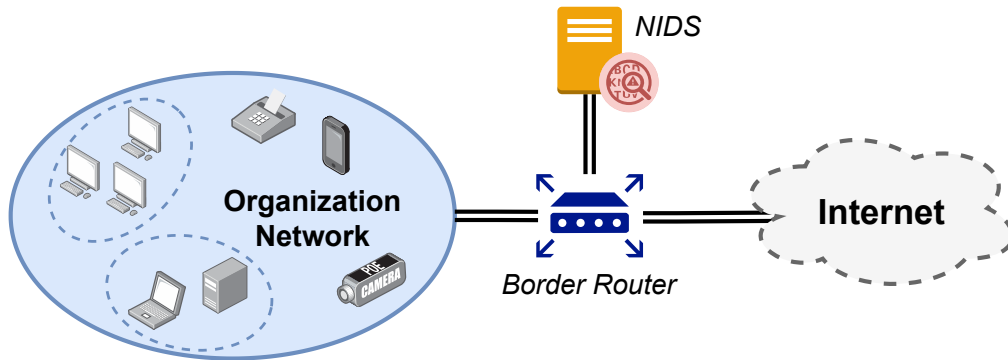


FIGURE 2.1: Typical NIDS scenario. The network of the organization can be composed of multiple subnetworks. The outgoing traffic passes through a border router which forwards such traffic to the internet, but also to a NIDS. The NIDS analyzes the traffic and, if necessary, raises some alerts.

model learns to make decisions from existing data. However, without some reference information, it is not possible to control what the ML model is actually ‘learning’ [144]. To specifically address detection (i.e., classification) problems, the training data must be separated into *benign* and *malicious* samples. In such circumstances, it is possible to develop autonomous ML-NIDS exploiting supervised ML methods. Such ‘supervision’ comes in the form of a human that must associate each sample in the training data to its ground truth, i.e., a *label* [269].

In some domains, labelling is simple (e.g., the popular captchas [42]) or labelled data can be used for a long period of time (e.g., ImageNet was collected in 2009 and is still widely used today [321]). However, the Cybersecurity domain is different: according to Miller et al. [203], a company can only label 80 malware samples per day. Specifically in NID, ground truth verification of network data is complex [204], and the concept drift problem requires any ML-NIDS to be continuously updated with new-labelled-data [143]. To aggravate this problem, deployment of any security system requires *proactive* evaluations conducted *in advance*, to avoid introducing a weak link in the security chain [28]. Hence, in the case of ML-NIDS, labelled data must be obtained both for the initial training, as well as for such evaluation.

The scarcity of labelled data for NID affected both research and practice [5], with an overall lack of ML-NIDS deployments, as well as a plethora of papers always based on the only publicly available dataset—the KDD99 [206].

2.1.2 Motivation: Mixing Network Data

The successes of ML renewed the interest of the NID community in these methods, and in recent years, many labelled datasets were made openly accessible (a survey is in [247]). However, most related work simply used such data as an ‘additional’ setting to perform their experiments. In contrast, in this paper we promote a different approach, based on mixing different network data to cross-evaluate ML-NIDS. Such opportunity, fostered by the recent availability of NID datasets, is of interest both for research and practice. Let us explain how mixing network data can assist ML-NIDS deployment. We first by present some high level applications (Section 2.1.2), and then provide a more specific use-case (Section 2.1.2).

Applications and advantages

Mixing data from different networks is useful to augment pre-existing datasets that contain an insufficient amount of labelled samples to develop ML-NIDS. It is also useful to assess the generalization capabilities of a ML-NIDS against ‘novel’ attacks not included in the training set (as very recently done by [234]). Such ‘novel’ samples can also be used extend the detection surface of the ML-NIDS by injecting them in the training set of the ML-NIDS. Similar strategies are particularly relevant to protect against the so-called ‘adversarial attacks’ which can evade traditional ML-NIDS [81]: the (new) training data can be leveraged for *adversarial training*, therefore realizing robust ML systems that can detect even subtle perturbations [1]. In this context, mixing diverse datasets facilitates the application of *ensemble* techniques (e.g., [324]), further increasing the resilience of ML-NIDS.

As stated by Biggio and Roli, empirical evaluations are always necessary for real deployments [31]. In this context, cross-evaluations are advantageous due to their low opportunity cost—especially when using publicly available data. Indeed, we observe that great attention has been given to *data sharing* platforms (e.g., [126]), and cross-evaluations could greatly benefit from dedicated ‘banks’ of NID data (e.g., [271]): it is true that the cybersecurity domain has high confidentiality, but anonymization techniques exists [240], and some recent solutions in *federated learning* overcame privacy issues (e.g. [75]). Finally, cross-evaluations can involve even *unsupervised* ML methods (e.g., anomaly detectors [88]), which represent the majority of currently deployed ML techniques for NIDS. Although unsupervised methods would not benefit from the ‘cheap’ labelling, they can still take advantage of the data diversity of different networks to assess (or improve) their generalization capabilities.

Exemplary use-case

Suppose an organization, \mathcal{O} , wants to protect their network, o , with a (supervised) ML-NIDS. Hence, \mathcal{O} collects and verifies some *benign* traffic data, N , from their network o . However, ML-NIDS also require *malicious* data, M . The following can happen w.r.t. such M :

- \mathcal{O} may not have any M generated in their network o . Hence, \mathcal{O} can ‘use’ some M generated in a different network than o – potentially of another organization.
- \mathcal{O} may have some M generated in o , obtained, e.g., by monitoring the behaviour of ‘known’ infected machines.

Therefore, \mathcal{O} can use such N and M to develop any ML model which, if it obtains appreciable performance, will be integrated in their security system as a ML-NIDS that can detect the attacks in M . Having an operational ML-NIDS, \mathcal{O} may be willing to assess whether such system can detect attacks not included in their M , which can potentially target the network o monitored by the ML-NIDS. To this end, \mathcal{O} can use a *small* set of malicious data originating from a network different than o , and containing different attacks than the ones ‘learned’ by their ML-NIDS. By using such malicious data to *evaluate* the ML-NIDS, \mathcal{O} can assess the generalization capabilities of their solution. If the assessment shows a weakness of the ML-NIDS, \mathcal{O} may acquire a *larger* set of such malicious data to extend the detection capabilities of their ML-NIDS, by using such data in the *training* stage. We will use the abovementioned example as basis for our demonstration in Section 2.5.

2.1.3 Related Work

The idea of cross-evaluating ML-NIDS on different datasets is not new. For instance, the authors of [44] propose a novel IDS dataset that can be used to evaluate the ‘transferability’ of ML-NIDS, but they do not provide any detailed analysis nor original experiment. Similarly, Pontes et al. [234] use a ML-NIDS trained on `IDS18` against `DDoS19`. However, [234] simply limit to *test* a novel method on a different dataset, and do not analyze the problem of ‘cross-evaluations’ as a whole, hence not allowing to highlight the benefits and limitations of such opportunity. For instance, cross-evaluations can also involve modifications of the *training* data, which is not covered by [234] and which is a case included in our demonstration.

Most prior works on ML-NIDS only assess their proposals in a single “context”, that is, the training and evaluation use the same dataset. For instance, the authors of [21] propose botnet detectors trained and tested on `CTU13`. In [299] a ML-NIDS focusing on different attacks is assessed on the `IDS18` dataset. To give a practical explanation, such methodology only allows determining that “the approach in [299] is effective on the network captured by the `IDS18` dataset, against the attacks contained in the `IDS18` dataset”. Other works may consider more datasets (e.g., [79, 135, 193, 255]), but the problem remains because the assessments are carried out independently on each dataset. Furthermore, all these works highlight that ML-NIDS require large (labelled) datasets—further motivating the need to explore novel solutions that mitigate the lack of labelled data. Among these, we mention *semisupervised* ML approaches (e.g. [327, 204]), which combine unlabelled with labelled data, and are hence orthogonal to our work.

A closely related research effort is [67], proposing a low-level software toolkit for analyzing NID datasets, forcing the user to abide to its constrained logic. For instance, it only works with data in the form of packet captures (PCAP), which require huge amounts of storage space and whose payload is often encrypted, making such data impractical to share (and, also, to analyze). In contrast, our proposed model is agnostic of the source data format (as long as there is some compatibility); moreover, our proposed framework operates on Network Flows (NetFlows), which represent a higher level than PCAP, making it flexible and extendible also to PCAP data—while not sacrificing performance [32, 21, 11].

We conclude that the idea of cross-evaluating ML-NIDS received only limited attention so far, and its opportunities and risks are still unknown. This is because no past research truly addressed such a problem—representing the core of this paper. Our intention is to provide a complete understanding of all the pros and cons related to cross-evaluations of ML-NIDS.

2.2 Modelling the Cross-evaluation of ML-NIDS

The intuition at the base of our work is to leverage *existing* NID datasets, with the goal of cross-evaluating ML-NIDS using samples from mixed networks. Such idea is grounded on the following observation (also implicitly adopted by [234]), which extends the well-known statement by Sommer et Paxson [269]: although every network is unique, the malicious behavior of network attacks is independent of the target network. For instance, Denial of Service (DoS) attacks always involve either a large amount of communications with minimal entity, or a smaller set of communications but with a larger entity – both happening in a short time frame [128]. Similarly, a machine infected by Botnet malware will periodically contact the CnC server, irrespective of what is happening in the ‘compromised’ network [14]. Hence,

such malicious behaviours can be used by a ML-NIDS to distinguish benign from malicious activities, regardless of the target network.

As the first effort to investigate this opportunity, we must design a *model* (Section 2.2.1) that allows to highlight its *benefits* (Section 2.2.2) as well as its intrinsic *challenges and risks* (Section 2.2.3).

2.2.1 Proposed Model Design

We now introduce all the prerequisites to describe our cross-evaluation model.

Let \mathbb{D} be a set of NID datasets which we denote as follows:

$$\mathbb{D} = (D_1, D_2, \dots, D_n), n \geq 2, \quad (2.1)$$

where n represents the cardinality of \mathbb{D} , and D_i represents a dataset collected in a given network i . Without loss of generality, we assume that each $D_i \in \mathbb{D}$ originates from a unique network environment—potentially, \mathbb{D} can include datasets representing distinct sub-networks within a larger network. Hence, in the remainder we use D_i and D_j ($i \neq j$) to denote two datasets of \mathbb{D} originating from two distinct networks (i.e., i and j). The *information* captured by each dataset in \mathbb{D} must allow one to use any subset of \mathbb{D} and derive a *set of common features* from such subset.²

Because our focus is on supervised ML for detection problems, each $D_i \in \mathbb{D}$ must be provided with *ground truth* distinguishing benign from malicious data. Hence, each dataset D_i can be seen as a composition of N_i , denoting the *benign* data of network i , and M_i , denoting the *malicious* data of network i . A dataset D_i can contain only malicious (or only benign) data; however, across all \mathbb{D} there must be at least a pair D_i, D_j for which $N_i \neq \emptyset$ and $M_j \neq \emptyset$. We denote with μ the number of *all* malicious classes contained in the entire \mathbb{D} . This is because any D_i can have a M_i with a variable number of attacks (i.e., Botnet, DoS, etc), which may overlap (or not) with those in a different M_j . Therefore, every M_i can be seen as an array of μ elements $M_i = (M_i^1, M_i^2, \dots, M_i^\mu)$, some of which can be empty if another dataset D_j has malicious classes not contained in D_i . Let \mathbb{N} denote the set of all benign samples, and \mathbb{M} denote the set of all malicious samples. We can visualize our model with the schematic in Figure 2.2, which shows the relationship between \mathbb{D} , \mathbb{N} and \mathbb{M} .

From Figure 2.2, we observe that all sets have n rows, each denoting a distinct source network dataset. However, while \mathbb{N} has only one column because all benign samples are treated equally, \mathbb{M} has μ columns representing all the attacks contained in \mathbb{D} . We provide an example in the caption of Figure 2.2. Such design makes our model suitable for $1+\mu$ classification ML problems, where a sample is either benign, or belongs to one among μ malicious classes. This automatically covers binary classification ML problems if all malicious classes are treated as a single malicious class (hence, $\mu=1$). Let us now use our proposed model to explain the benefits brought by cross-evaluations of ML-NIDS.

2.2.2 Benefits: additional Contexts

Deployment of ML components requires a *training* set T , used to develop a ML model, and an *evaluation* set E , used to assess the performance of such model. Hence,

²For instance, it is possible that D_i comes as PCAP traces, and D_j comes as NetFlows: in this case, the PCAP of D_i can be processed to derive the NetFlows features of D_j . Similarly, two datasets D_i and D_j can contain NetFlows generated with different software: in this case, the features shared by D_i and D_j can represent the common set.

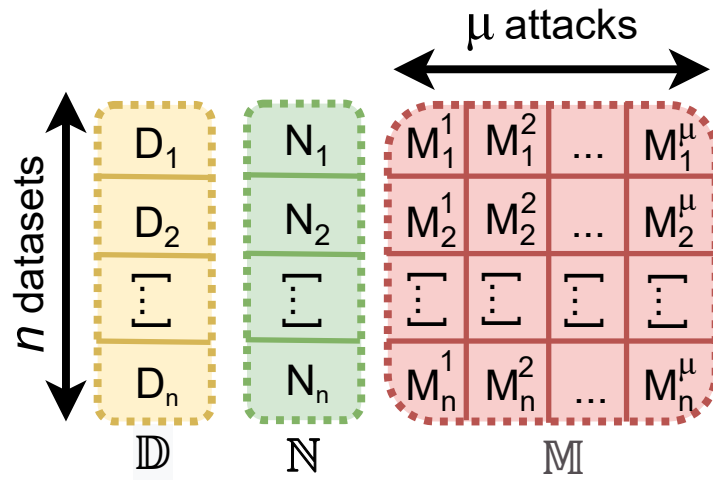


FIGURE 2.2: Proposed cross-evaluation model, representing \mathbb{D} , \mathbb{N} and \mathbb{M} . **Example:** assume that \mathbb{D} has three datasets: D_1, D_2, D_3 (hence $n=3$). Assume that: D_1 has some benign samples but no malicious samples; D_2 has no benign samples, but malicious samples of a *botnet* attack, and also some malicious samples of a *portscan* attack; D_3 has some benign samples, alongside some malicious samples of the same *botnet* attack as D_2 , but also some malicious samples of a *DoS* attack. In this case, \mathbb{N} will contain three elements: N_1, N_2, N_3 , with N_2 being empty. On the other hand, there are three malicious classes ($\mu=3$) hence \mathbb{M} will have nine elements: M_2^1 and M_3^1 (the botnet attack shared by D_2 and D_3) as well as M_2^2 (the portscan attack in D_2) and M_3^3 (the DoS attack in D_3) will contain some samples; whereas the remaining will be empty (i.e., all those with M_1 , as well as M_2^3 and M_3^2).

our idea is composing T and E by drawing from \mathbb{N} and \mathbb{M} : depending on the draw, a specific ‘context’ is created that can be used to cross-evaluate a ML-NIDS. The main benefits provided cross-evaluations of ML-NIDS are due to the increased types of contexts that can be assessed, which can be highlighted with our proposed model.

Because our model is rooted on Sommer and Paxson statement (Section 2.2), it is crucial that both T and E use benign samples from the same network³, which should represent the environment where the ML-NIDS is to be deployed; hence, let o (standing for ‘origin’) denote such network, and N_o be the corresponding benign samples. Then, we observe that there are many ways to compose T and E by choosing the malicious element from \mathbb{M} . Such variability can be modeled through the ‘matrix’ \mathbb{M} in Figure 2.2, by pinpointing which rows and columns are included in T and E . The following can occur:

- T (or E) can contain malicious samples either from the same or different o (i.e., same or different row than N_o);
- the malicious samples in T and E can come either from the same or different networks (regardless of N_o);
- the malicious class(es) in T can be either the same or different than those in E (i.e., same or different columns).

In particular, let t and e denote two rows of \mathbb{M} ; and let τ and ε denote two columns of \mathbb{M} ; we use such notation to identify two elements of \mathbb{M} , i.e., M_t^τ and M_e^ε .

Let \vec{t} , \vec{e} , $\vec{\tau}$, $\vec{\varepsilon}$ be four *ordered arrays*⁴, each denoting multiple columns or rows (e.g., \vec{t} contains multiple t , i.e., rows) of \mathbb{M} . Let \vec{o} be an unary array including only o (representing the benign ‘origin’ network from \mathbb{N}).

By following such notation, we can represent the training and evaluation sets, T and E , as the functions in Expression 2.2:

$$T(\vec{o}, \vec{t}, \vec{\tau}), \quad E(\vec{o}, \vec{e}, \vec{\varepsilon}). \quad (2.2)$$

Simply put, T and E are denoted by a single row of \mathbb{N} (i.e., \vec{o}), and the rows and columns of all the elements of \mathbb{M} that they include (i.e., \vec{t} and $\vec{\tau}$ for T , while \vec{e} and $\vec{\varepsilon}$ for E).

We can see a *context* as a function of T and E . Specifically, a context C is denoted as the following tuple:

$$C(T, E) \Rightarrow C(\vec{o}, \vec{t}, \vec{e}, \vec{\tau}, \vec{\varepsilon}). \quad (2.3)$$

Depending on the elements from \mathbb{N} and \mathbb{M} included in T and E , many contexts can be reproduced, which can be of different *type*. In particular, let \vec{o} , \vec{h} , \vec{e} , $\vec{\tau}$, $\vec{\varepsilon}$ denote the *sets* of the corresponding arrays (each element of a given set is unique). By cross-evaluating ML-NIDS, it is possible to assess 10 different context types, which are denoted by the relationships between \vec{o} , \vec{h} , \vec{e} , $\vec{\tau}$, $\vec{\varepsilon}$.

We provide the full list of such context types in Table 2.1; we also include a practical example in the caption of Table 2.1. Specifically, for each context type (denoted with a number after the letter C), we report the four conditions denoting the relationships among all the involved components; on the same line of each condition, we describe the consequences on T and E ; we also provide a concrete use case that

³This also serves to reduce false alarms *after* potential deployments, because the benign samples always have the same source.

⁴Such arrays can be of variable length, but $|\vec{t}|=|\vec{\tau}|$ and $|\vec{e}|=|\vec{\varepsilon}|$ must be true.

explains the application of such type of context. We note that all cases where two sets are not equal can be further split in two: when one set is a *superset* of the other; and when they are *disjointed*.

TABLE 2.1: The 10 types of contexts enabled by the proposed cross-evaluation model. The table represents the relationship among the unique elements of \mathbb{N} and \mathbb{M} included in the training and evaluation sets, T and E (cf. Expression 2.2), which denote a context (cf. Expression 2.3). The origin of benign samples is always the same in T and E , hence \bar{o} is shared. Contexts of type C1, C2 are those considered by most prior works; C4 has been considered in [234]. A gray background denotes the types of context assessed in our demonstration. **Example:** consider the setting described in the example of Figure 2.2, where $n=3$ and $\mu=3$. Suppose a context denoted by: $\bar{o}=(3)$, $\bar{t}=(2,3)$, $\bar{e}=(3)$, $\bar{\tau}=(1,1)$, $\bar{\epsilon}=(1)$. Such context implies that T contains benign samples from N_3 , and malicious samples from M_2^1 and M_3^1 ; whereas V contains benign samples from N_3 but malicious samples from M_3^1 . The resulting context will be of type C5, because $\bar{o}=(3)$, $\bar{t}=(2,3)$, $\bar{e}=(3)$, $\bar{\tau}=(1)$, $\bar{\epsilon}=(1)$. Hence, $(\bar{o} = \bar{e}) \neq \bar{t}$ and $\bar{\tau} = \bar{\epsilon}$.

| C-type | Conditions | Effects on T and E | Use-case |
|--------|--|---|---|
| C1 | $\bar{o} = \bar{t}$ $\bar{o} = \bar{e}$ $\bar{t} = \bar{e}$ $\bar{\tau} = \bar{\epsilon}$ | T uses benign and malicious samples from the same network. E uses benign and malicious samples from the same network. T and E use malicious samples from the same network. T and E use the same attack classes. | This is the standard 'baseline' case commonly used in research. |
| C2 | $\bar{o} = \bar{t}$ $\bar{o} = \bar{e}$ $\bar{t} \neq \bar{e}$ $\bar{\tau} \neq \bar{\epsilon}$ | T uses benign and malicious samples from the same network. E uses benign and malicious samples from the same network. T and E use malicious samples from the same network. T and E use different attack classes. | Same as the baseline C1 but the ML-NIDS is tested on different attacks. |
| C3 | $\bar{o} = \bar{t}$ $\bar{o} \neq \bar{e}$ $\bar{t} \neq \bar{e}$ $\bar{\tau} = \bar{\epsilon}$ | T uses benign and malicious samples from the same network. E uses benign and malicious samples from different networks. T and E use malicious samples from different networks. T and E use the same attack classes. | Using the ML-NIDS on the same attacks, but targeting hosts from a different network. |
| C4 | $\bar{o} = \bar{t}$ $\bar{o} \neq \bar{e}$ $\bar{t} \neq \bar{e}$ $\bar{\tau} \neq \bar{\epsilon}$ | T uses benign and malicious samples from the same network. E uses benign and malicious samples from different networks. T and E use malicious samples from different networks. T and E use different attack classes. | Testing the generalization capabilities of the NIDS on completely unknown attacks (different hosts, different attacks). |
| C5 | $\bar{o} \neq \bar{t}$ $\bar{o} = \bar{e}$ $\bar{t} \neq \bar{e}$ $\bar{\tau} = \bar{\epsilon}$ | T uses benign and malicious samples from different networks. E uses benign and malicious samples from the same network. T and E use malicious samples from different networks. T and E use the same attack classes. | When there are too few samples of an attack class to both train and test a ML-NIDS, it is possible to borrow some malicious samples from another network and use them in T . |
| C6 | $\bar{o} \neq \bar{t}$ $\bar{o} = \bar{e}$ $\bar{t} \neq \bar{e}$ $\bar{\tau} \neq \bar{\epsilon}$ | T uses benign and malicious samples from different networks. E uses benign and malicious samples from the same network. T and E use malicious samples from different networks. T and E use different attack classes. | Assessing whether training on new attacks (different hosts and attack class) affects the detection on 'known' attacks. |
| C7 | $\bar{o} \neq \bar{t}$ $\bar{o} \neq \bar{e}$ $\bar{t} = \bar{e}$ $\bar{\tau} = \bar{\epsilon}$ | T uses benign and malicious samples from different networks. E uses benign and malicious samples from different networks. T and E use malicious samples from the same network. T and E use the same attack classes. | Extending the detection surface of the ML-NIDS by training on attacks originating from different networks and testing such ML-NIDS against these attacks. |
| C8 | $\bar{o} \neq \bar{t}$ $\bar{o} \neq \bar{e}$ $\bar{t} = \bar{e}$ $\bar{\tau} \neq \bar{\epsilon}$ | T uses benign and malicious samples from different networks. E uses benign and malicious samples from different networks. T and E use malicious samples from the same network. T and E use the different attack classes. | Testing an 'extended' ML-NIDS devised by exploiting C7 against other attacks for which not enough samples are available to train a dedicated detector. |
| C9 | $\bar{o} \neq \bar{t}$ $\bar{o} \neq \bar{e}$ $\bar{t} \neq \bar{e}$ $\bar{\tau} = \bar{\epsilon}$ | T uses benign and malicious samples from different networks. E uses benign and malicious samples from different networks. T and E use malicious samples from different networks. T and E use the same attack classes. | Using the benign samples of a different network and combining them with 'owned' malicious samples whether the 'owned' malicious samples |
| C10 | $\bar{o} \neq \bar{t}$ $\bar{o} \neq \bar{e}$ $\bar{t} \neq \bar{e}$ $\bar{\tau} \neq \bar{\epsilon}$ | T uses benign and malicious samples from different networks. E uses benign and malicious samples from different networks. T and E use malicious samples from different networks. T and E use different attack classes. | Training a ML-NIDS using benign and malicious sample from two distinct subnets, and testing the generalization capabilities of such ML-NIDS against other attacks targeting a different subnet. |

Past works (e.g., [11, 274]) only considered cases where the 'row' was fixed, i.e., where $\bar{o} = \bar{t} = \bar{e}$, corresponding to the contexts of type C1 and C2. Pontes et al. [234] investigated C4. In contrast, it is evident from Table 2.1 that our cross-evaluation model enables the assessment of 7 additional context types, allowing to discern additional qualities of ML-NIDS and corresponding NID datasets. For instance, all the scenarios envisioned in our motivational example (cf. Section 2.1.2) can be represented by the context types listed in Table 2.1.

In our demonstration (Section 2.5), we first assess C1 as ‘baseline’ comparison with the state-of-the-art; and then we consider C4 (as done by [234]), and C7 (the latter both in its ‘disjoined’ and ‘extended’ variants). We observe that our cross-evaluation idea is different from the practice of ‘transferring ML models’ (common in Computer Vision [217]): in our idea, the benign data always comes from the same network. The reason becomes apparent in the next section.

2.2.3 Challenges and Risks

The cross-evaluation of ML-NIDS has high potential, but a superficial application can lead to dangerous consequences—spanning from underwhelming performance to additional security risks. Indeed, mixing data from different networks presents several fundamental issues, which must be known when real ML-NIDS deployments are considered. We stress that our paper lies at the intersection of diverse research fields (i.e., network traffic analysis, machine learning, cybersecurity) and some of the following issues may be well-known within each field. Considered the scope of our paper, it is meaningful to make the entire community aware of such issues.

We identify the following three performance-related implementation *challenges*:

1. *Removing Network Artifacts*. Depending on the considered set of features, some samples may contain ‘artifacts’ that are unrelated to their benign/malicious nature⁵. If not sanitized, such artifacts may be learned by the ML model to perform its decisions, leading to overfitting and, hence, useless ML-NIDS.
2. *Preserving Performance*. When a given context involves modifications of T , it is important not to degrade the baseline False Positive Rate (FPR). Modifications of T must always be assessed.
3. *Maximizing Performance*. Assuming that simply adding malicious samples to T results in a ML-NIDS capable of detecting such attacks is misleading: it has been shown that ML models for NIDS may yield underwhelming detection performance in multi-classification settings [11]. It is hence crucial to consider a ML-NIDS architecture that optimizes the usage of such additional samples.

Finally, we highlight three intrinsic *risks* that involve security aspects of cross-evaluations of ML-NIDS.

- *Labeling quality*. Cross-evaluations are significant only if the samples in \mathbb{N} and in \mathbb{M} all report the correct label (i.e., benign or malicious). If, e.g., \mathbb{N} contains malicious samples because the authors of the source dataset did not perform proper verifications, then the final results may be unreliable. Unless the cross-evaluation involves unsupervised ML, real deployments should ensure that all samples are associated to the correct ground truth.
- *Exposure to adversarial ML attacks*. Although mixing data from different networks can result in more resilient ML-NIDS (cf. Section 2.1.2), relying on *public* datasets exposes to ‘poisoning’ attacks. In these circumstances, training a ML-NIDS on such data would have the opposite effect of adversarial training. For instance, in [84] the FPR increases by 5 times when only 5% of the data

⁵The most blatant example is when a dataset has all its malicious samples originating from the same IP address. If the IP address is considered as a feature, the ML model will only look for the ‘malicious’ IP address, meaning that any attack involving other machines will never be detected.

is polluted. More subtle poisoning strategies exploit ‘backdoors’ which make ML-NIDS prone to evasion, as evidenced in [20]—and also in [215] for federated learning scenarios. Countermeasures include verifying the checksum of each dataset as provided by the authors; or applying some modifications that can remove or mitigate the effects of such poisoned samples (e.g., [13]).

- *Incompatible Networks.* Regardless of the resulting performance, mixing samples from different networks may not be possible a-priori. If the goal is using an N from a different network, it is necessary to conduct *preliminary* analyses ensuring that the two networks are indeed similar. On the other hand, when using M from different networks it is necessary to perform *follow-up* analyses that question the validity of the cross-evaluation results. This is because high detection rates at test-time may lead to a ‘false sense’ of security if the malicious activities depend on the underlying network’s behaviour. Such analyses may include comparing the feature importance between different models; or complete sanity checks by deploying the ML-NIDS against true attacks—in real time. Regardless, using the same source of benign samples both in T and E ensures that the FPR after deployment will not deviate from the one at test-time.

We can conclude that the additional context types enabled by cross-evaluations of ML-NIDS are intriguing, but practical applications are not simple and require the adoption of a rigorous workflow.

2.3 Proposed Framework: XeNIDS

We showed that cross-evaluations of ML-NIDS are enticing but challenging, and we are not aware of efforts that tackled this problem in an exhaustive way. As a first step, we propose XeNIDS, a framework for the Cross-evaluation of Network Intrusion Detection Systems based on machine learning, with a focus on NetFlow data.

Our proposed framework is rooted in the same design principles described in Section 2.2.1, and has a threefold goal:

- allowing the *simulation* of all contexts in Table 2.1;
- facilitating assessments of *multiple* contexts;
- addressing the challenges discussed in Section 2.2.3.

Of course, we do not claim that XeNIDS is the only way to do all of the above. Our intention is to further promote the diffusion of cross-evaluations in *research*, as well as to increase their realistic value for proactive assessments.

We provide an overview of XeNIDS in Section 2.3.1, which consists in four stages: *standardize* (Section 2.3.2), *isolate* (Section 2.3.3), *contextualize* (Section 2.3.4), *cross-evaluate* (Section 2.3.5).

2.3.1 Overview

The focus of XeNIDS is on NetFlows (Section 2.1.3), enabling *inter-compatibility* with PCAP data. NetFlows are metadata generated from packet captures, and summarize the communications between two endpoints. A NetFlow is defined as:

$$\text{NetFlow} = (\text{srcIP}, \text{dstIP}, \text{srcPort}, \text{dstPort}, t, \text{proto}, d, \dots), \quad (2.4)$$

where $srcIP$ ($srcPort$) and $dstIP$ ($dstPort$) are the source and destination IP addresses (network ports) of the two involved hosts, t is the timestamp of the first connection, d is the duration of the communication session, $proto$ is the network protocol of the communication. Depending on the NetFlow software and its configuration, additional metrics can be computed: the most typical fields include the number of *packets* and *bytes* exchanged during the communication [303].

We present a schematic representation of XeNIDS in Figure 2.3. XeNIDS requires

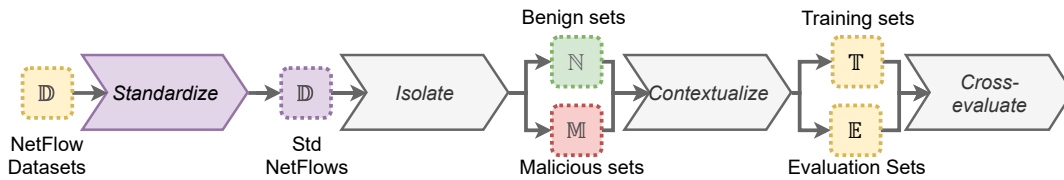


FIGURE 2.3: Overview of XeNIDS. The input \mathbb{D} is a set of labelled NetFlow datasets of n distinct networks. The output results should be further analyzed.

a set of n datasets containing NetFlows, representing \mathbb{D} and totalling μ distinct attack classes. These datasets must be provided with the *ground truth*. XeNIDS assumes that all data in \mathbb{D} is verified, trusted and appropriate for the considered deployment scenario (Section 2.2.3).

The framework includes four stages (cf. Figure 2.3):

1. *Standardize*: the input datasets in \mathbb{D} are first cleaned and sanitized, and then brought into a common ‘language’.
2. *Isolate*: every standardized dataset is partitioned in its benign and malicious sets (\mathbb{N} and \mathbb{M}).
3. *Contextualize*: \mathbb{M} and \mathbb{N} are used to compose a context (by generating the corresponding \mathbb{T} and \mathbb{V}).
4. *Cross-evaluate*: \mathbb{T} and \mathbb{V} are used to develop and cross-evaluate a ML-NIDS.

The results provided as *output* by XeNIDS should be *further analyzed* for practical deployments (cf. Section 2.2.3).

2.3.2 Standardize

In the first stage, schematically depicted in Figure 2.4, XeNIDS brings all the datasets $D_i \in \mathbb{D}$ into a common NetFlow format, accounting for potential obfuscations as a result of *anonymization* techniques. Essential operations involve data sanitization (e.g., handling missing values) and filtering: for example, if the goal is the detection of attacks involving TCP traffic, then all non-TCP traffic can be safely removed. Then, the focus is on establishing a common feature set⁶ while simultaneously *removing network artifacts* that may lead to overfitting. Such procedures are tough, especially when considering NetFlow records, but necessary. To explain the reasons of such difficulties and our proposed workarounds, we provide an *original interpretation* of NetFlows with respect to machine learning.

In simple terms, a NetFlow is the result of two contributors: the *communications* (*Comm*) performed by the involved hosts, and the effects of the *environment* (*Env*)

⁶Taken from the *intersection* of the features across all \mathbb{D} .

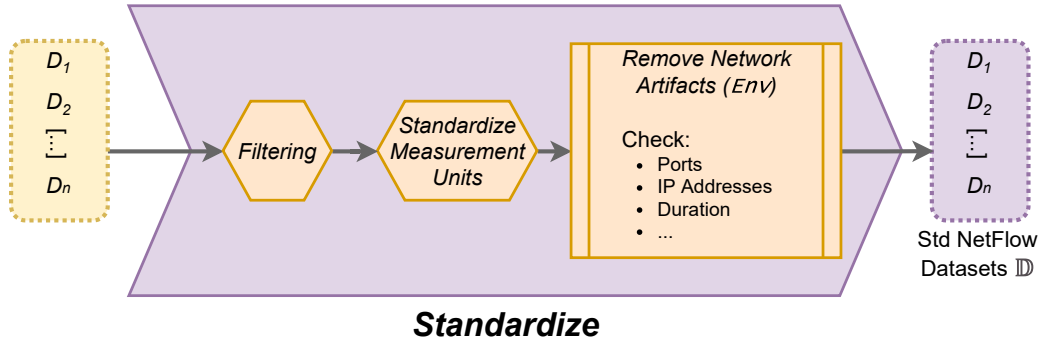


FIGURE 2.4: First stage: Standardize. The initial NetFlows in \mathbb{D} are all standardized to derive a common feature set, and cleaned of any possible artifact that may lead to overfitting and impractical ML-NIDS.

where the NetFlow is generated. This latter factor (Env) is, in turn, influenced by two elements: the network *identity* ($NetId$), denoting the intrinsic characteristics of the network where $Comm$ (such as allocated bandwidth, protocols used, common open ports, periodic services) are captured; and the *configuration* of the appliance ($Conf$) used to generate the NetFlows. Hence, the information captured by a NetFlow is a function⁷ \mathcal{F} of three components: $Comm$, $NetId$, $Conf$. Formally:

$$\mathcal{F}(Comm, Env) \Rightarrow \mathcal{F}(Comm, NetId, Conf) = \text{NetFlow} \quad (2.5)$$

The ultimate goal of the standardize stage is to mitigate the effects of Env (represented by $NetId$ and $Conf$) across all the input datasets in \mathbb{D} . Indeed, if one dataset D_i has an Env that is significantly stronger than D_j , then a ML model trained on data from D_i and D_j may only learn on the basis of such ‘signature’ Env . These circumstances lead to overfitting on Env , resulting in impractical detectors that neglect to search for malicious behaviours.

Let us explain Expression 2.5 with two practical use-cases on the contribution of Env .

- Different $NetId$. Consider two different networks where a host downloads the same file from the same remote server via SSH: if these two networks use different listening ports for the SSH server (e.g., 22 and 4022), then the NetFlows of the first network will differ from those of the second network (they will have different ports).
- Different $Conf$. Using different NetFlow software and/or settings yields different NetFlows even when the original PCAP traces are identical. For instance, measurement units can differ, resulting in datasets that are not compatible: a dataset D_i with d expressed in milliseconds cannot be used alongside a dataset D_j that uses seconds.

We report in Appendix 2.8.1 an exhaustive explanation of the effects brought by Env on NetFlows.

By referring to the official NetFlow v9 documentation⁸, we observe that there are several fields that can contribute to Env (influenced both by $NetId$ and $Conf$), which require particular care at this stage. We provide in Appendix 2.8.2 some recommendations for reducing the generation of the above-mentioned artifacts, with a focus on three fields: the IP address, the network ports, and the flow duration.

⁷The definition of \mathcal{F} is software dependent [303], and outside our scope.

⁸www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/

Nonetheless, depending on the considered use-cases, many low-level implementations are viable to minimize the impact of *Env* and derive a common feature set. After this stage, the initial set of datasets \mathbb{D} is standardized and ready for the ‘core’ functionalities of XeNIDS.

2.3.3 Isolate

In this stage, XeNIDS isolates the benign from malicious samples of each (standardized) dataset in \mathbb{D} to derive \mathbb{N} and \mathbb{M} (cf. Figure 2.2 in Section 2.2.1). We provide a schematic in Figure 2.5.

Specifically, XeNIDS first partitions the benign from malicious samples in each D_i , resulting in two distinct sets, N_i and M_i . Then, XeNIDS further partitions the specific attack samples in M_i according to the *individual* attack that they represent⁹ (assuming that $\mu > 1$).

Such design choice enables the development of *collaborative ensembles* (e.g., [234, 2]) of ML classifiers, each devoted to a specific threat, therefore addressing the third challenge (cf. Section 2.2.3)—while also allowing to use XeNIDS for multi-classification ML problems.

We note that the separation can also account for a specific level of *granularity*. In this case, the original μ will be changed by ‘aggregating’ attacks of different classes, potentially even treating all malicious samples as belonging to a single malicious class. Depending on the use-case, such granularity can vary: it could either be performed at a high-level (e.g., *Botnet* or *DoS* attacks) or go at a deeper level (e.g., a specific *Botnet* variant). The final choice depends on the actual use-case (e.g., when there are not enough samples available, they can be aggregated into a macro-class, or simply discarded).

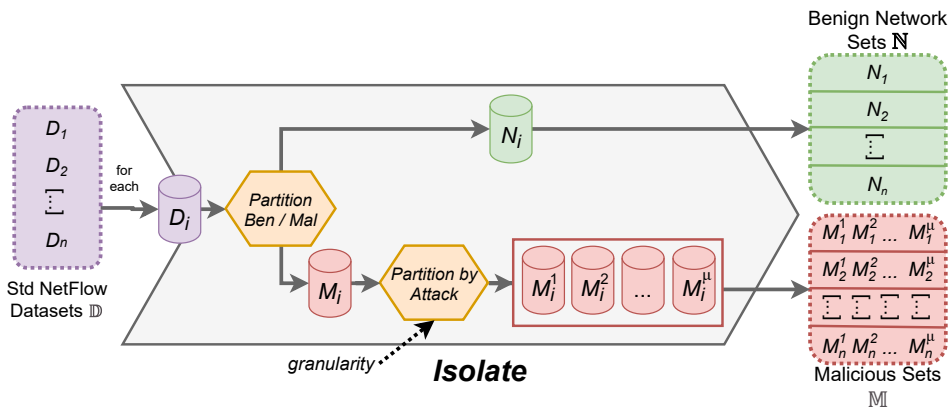


FIGURE 2.5: Second stage: isolate. The standardized \mathbb{D} is used to extract the n individual benign N and malicious M sets. The latter are then split in μ sets on the basis of their attack class. Depending on the user-provided granularity, it is possible to aggregate sets of different attack classes into a bigger class, therefore changing the initial μ . The outputs are the full sets \mathbb{N} and \mathbb{M} .

This stage produces two outputs: \mathbb{N} , containing all the benign network samples (partitioned in n subsets according to their source dataset); and \mathbb{M} , containing all malicious samples isolated in $n \times \mu$ subsets of samples according to their specific attack and source dataset. We recall that some elements of \mathbb{M} can be empty, i.e., if a D_i does not contain malicious samples of the same classes as D_j .

⁹The attack is denoted by the ground-truth labels provided in the input \mathbb{D} .

2.3.4 Contextualize

In the third stage, XeNIDS creates *all* the sets corresponding to the contexts to simulate during the cross-evaluation. This is done by using \mathbb{N} and \mathbb{M} (provided by the previous stage), alongside some external input, to compose training T and evaluation E sets; all such T and E will be put in two dedicated collections, \mathbb{T} and \mathbb{E} . We provide a schematic of this stage in Figure 2.6.

Two user-provided input *lists* regulate this stage: a 5-dimensional tuple of context-related parameters $(\vec{o}, \vec{t}, \vec{e}, \vec{\tau}, \vec{\epsilon})$; and a pair of *splits*, $s(N)$ and $s(M)$, used to partition (e.g., 80:20) any N and M that will be included in a given T and E . The idea is to facilitate cross-evaluations that consider multiple contexts, by composing all the necessary T and E *before* using them for any assessment. Hence, XeNIDS iterates over all the elements in the two input lists: at each iteration, XeNIDS composes a training and evaluation set according to the user-specified parameters.

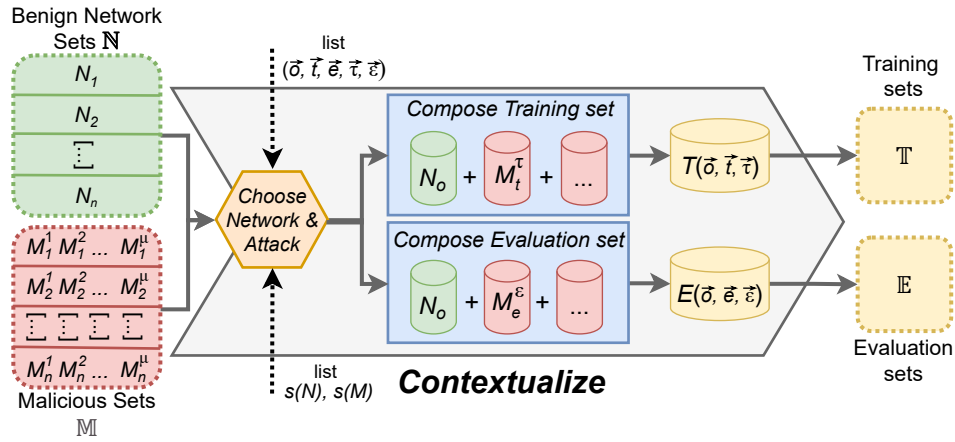


FIGURE 2.6: Third stage: contextualize. XeNIDS uses \mathbb{N} and \mathbb{M} to create multiple T and E , according to the context-related parameters $(\vec{o}, \vec{t}, \vec{e}, \vec{\tau}, \vec{\epsilon})$ and the splits $s(N)$ and $s(M)$. All the composed T and E are inserted in \mathbb{T} and \mathbb{E} . **Example.** Assume that the following user-provided parameters: $\vec{o}=(1)$, $\vec{t}=(2)$, $\vec{e}=(3)$, $\vec{\tau}=(1)$, $\vec{\epsilon}=(1)$; $s(N)=(80:20)$, $s(M)=(70:30)$. XeNIDS chooses N_1 , and puts 80% of N_1 in T and the remaining 20% in V . XeNIDS then selects M_2^1 and puts 70% of its samples in T ; then XeNIDS selects M_3^1 and puts 30% of its samples in V . Such T (and V) is then inserted in \mathbb{T} (and \mathbb{V}). The operation is repeated if the user provides additional lists as input.

Specifically, for each tuple $(\vec{o}, \vec{t}, \vec{e}, \vec{\tau}, \vec{\epsilon})$, and for each pair of splits, $s(N)$ and $s(M)$, XeNIDS proceeds as follows.

- XeNIDS uses o to select a specific set of benign samples N_o from \mathbb{N} . XeNIDS splits N_o according to $s(N)$, and puts the corresponding partitions in T and E .
- For each $(t, \tau) \in (\vec{t}, \vec{\tau})$, XeNIDS extracts from \mathbb{M} the element M_t^τ , which is split with $s(M)$ and put in T .
- For each $(e, \epsilon) \in (\vec{e}, \vec{\epsilon})$, XeNIDS extracts from \mathbb{M} the element M_e^ϵ , which is split with $s(M)$ and put in E .
- Alltogether, these operations result in two sets, $T(\vec{o}, \vec{t}, \vec{\tau})$ and $E(\vec{o}, \vec{e}, \vec{\epsilon})$, which are put in \mathbb{T} and \mathbb{E} .

An example of such workflow is in the caption of Figure 2.6. In cases where $t=e$ and $\tau=\varepsilon$, XeNIDS performs the partitioning simultaneously, to avoid overlaps that can result in the same malicious samples being included in both T and E .

The selection of $s(N)$ and $s(M)$, which can differ for T and E , must be done to achieve a twofold goal: (i) realize an E that is comprehensive enough to cover the real data distribution, and hence produce insightful results; and (ii) realize a T that allows to develop proficient ML-NIDS. For instance, if E does not contain many benign samples, then the resulting FPR may not correspond to the real FPR after the ML-NIDS is deployed. At the same time, if T contains only a small number of samples for a given attack, the resulting ML-NIDS will not be able to capture all the possible variations of such attack¹⁰. After this stage, we obtain two collections of training and evaluation sets, \mathbb{T} and \mathbb{E} .

2.3.5 Cross-evaluate

In the last stage, XeNIDS performs the cross-evaluation by using the sets in \mathbb{T} and \mathbb{E} to reproduce any user-specified context. Hence, the input parameters are a list of contexts C (cf. Expression 2.3); as well as a learning ML algorithm to develop the detectors of the ML-NIDS.

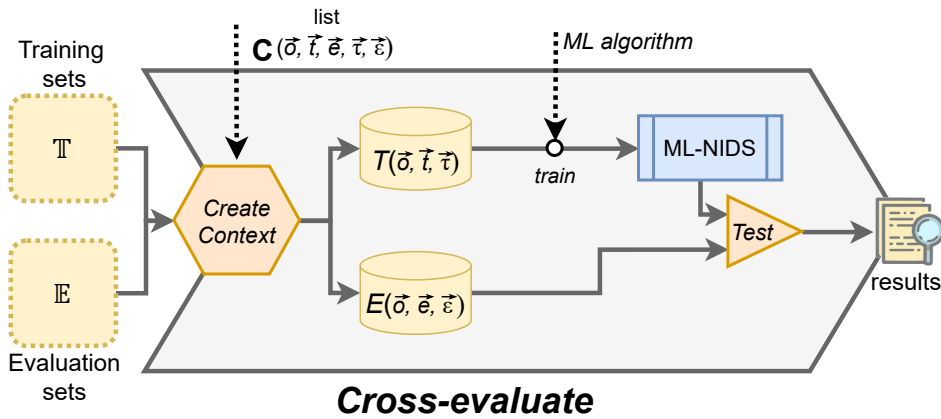


FIGURE 2.7: Final stage: cross-evaluate. XeNIDS reproduces the user-specified context(s) and performs the cross-evaluation. **Example:** assume the following input context: $C((1), (1,2), (1,2), (1,1), (2,2))$ and a ML-NIDS that leverages ensembles of binary detectors. XeNIDS first extracts $T((1),(1,2),(1,1))$ from \mathbb{T} , which is split in two smaller sets, $T(1,1,1)$ and $T(1,2,1)$. Such sets are used to train two ML-models that will compose the ML-NIDS. The ML-NIDS can then be tested either against $E((1),(1,2),(2,2))$, or against its subsets $E(1,1,2)$ and $E(1,2,2)$, all of which obtained from \mathbb{E} . Any previously trained model (e.g., the one using $T(1,1,1)$) can be reused to assess different contexts.

Specifically, for each context $C(\bar{\sigma}, \bar{\tau}, \bar{\varepsilon}, \bar{\tau}, \bar{\varepsilon})$ provided as input, XeNIDS draws the corresponding $T(\bar{\sigma}, \bar{\tau}, \bar{\tau})$ and $E(\bar{\sigma}, \bar{\varepsilon}, \bar{\varepsilon})$ from \mathbb{T} and \mathbb{E} . Then, depending on the architecture of the ML-NIDS, XeNIDS operates as follows.

- If the ML-NIDS leverages a *single classifier*, XeNIDS uses $T(\bar{\sigma}, \bar{\tau}, \bar{\tau})$ to train a single (multi-class) ML-model with a given ML algorithm; such ML-model is then tested against $E(\bar{\sigma}, \bar{\varepsilon}, \bar{\varepsilon})$.

¹⁰We refer the reader to [327] for a study on how the size of the training set can impact the performance of ML-NIDS.

- If the ML-NIDS leverages *ensembles of classifiers*, XeNIDS splits $T(\vec{o}, \vec{t}, \vec{\tau})$ into smaller sets, (e.g., by composing $T(o, t, \tau)$ focusing on the specific attack τ contained in t); each of these sets is used to train a dedicated ML-model of the ensemble. Such procedure can be repeated for $E(\vec{o}, \vec{e}, \vec{\epsilon})$, i.e., the ML-NIDS can be tested against the entire E , or against subsets.

The design of XeNIDS enables the assessment of multiple contexts without the need of training additional ML models. If two contexts require the same T , it is only necessary to draw a different E from \mathbb{E} , and use such E to assess the previously trained ML-NIDS.

We illustrate this stage in Figure 2.6, where we also provide a complete example of an ensemble use-case. We anticipate that, in our demonstration, we will always use ensembles of specialized classifiers.

The results produced as output of this stage should be subject to subsequent analyses and considerations.

2.4 Application

As a final contribution of this paper, we showcase¹¹ a practical application of XeNIDS. We do so via a large set of experiments where we cross-evaluate ML-NIDS by using a total of 6 well-known NID datasets.

We describe our testbed (Section 2.4.1) and explain the preprocessing operations (Section 2.4.2). Then, we present the common assessment procedure (Section 2.4.3).

2.4.1 Testbed

The aim of our demonstration is reproducing and assessing the use-cases described in Section 2.1.2. To this purpose, we assess three different context types (cf. Table 2.1), namely C1, C4 and C7. However, we differentiate our experiments depending on the *format* of the NetFlow data used as input to XeNIDS: specifically, such data can be either in a *uniform* or *heterogeneous* format. Let us explain our rationale and the differences between these two distinct scenarios.

We recall that XeNIDS operates on *existing* data in the form of NetFlows. Such NetFlows can be provided either (a) as PCAP traces, and then exported to NetFlows using dedicated software; or (b) directly as NetFlows, processed according to the creators' specifications. These two scenarios must be treated separately, due to the different effects that they can have on the *results*. In the first scenario, the raw PCAP traces (collected in diverse network environments) can be used to generate *uniform* NetFlows by using the same appliance for all PCAP traces; because the NetFlows share the same format, the results are more reliable due to a lower chance of network artifacts (the contribution of *Conf* is the same—cf. Expression 2.5). However, such scenario requires *all* source data to be fully provided as PCAP, which is a requirement that is hard to meet¹². Therefore, it is insightful to consider also the scenario where the source datasets are provided directly as *heterogeneous* NetFlows (due to being generated with different software). Such scenario requires a more careful application of XeNIDS's *standardize* stage (Section 2.3.2), but also a more detailed analysis of the results because the effects of different initial *Conf* can only be seen *after* the ML-NIDS is evaluated.

¹¹Our implementation of XeNIDS: <https://github.com/pajola/XeNIDS>

¹²E.g., PCAP data can be truncated [101] or not fully labelled [263].

We hence apply XeNIDS differently for both scenarios, each considering 4 well-known datasets ($n=4$ for both scenarios).

- *Heterogeneous* scenario: here, we use the CTU13 [101], NB15 [210], IDS18 [262], DDOS19 [263]. These are all provided as NetFlows, but using different appliances.
- *Uniform* scenario: here, we use the UF-BotIoT, UF-NB15, UF-IDS18, UF-ToNIoT. These datasets are created in [255] by the PCAP version of existing datasets and generating the corresponding (labelled) NetFlows using a unified appliance.

For comparison purposes, two datasets are shared¹³, whereas two are unique for each scenario.

We provide in Table 2.2 an overview of these datasets. For each dataset, we report the amount of NetFlows, the overall number of malicious classes, the size of the provided feature-set, and the performance (as F1-score) achieved by the state-of-the-art. From this table, we can already observe the effects of *Conf* on the corresponding NetFlows: two datasets (i.e., the NB15 and the IDS18) are used by both scenario, but the amount of samples and features differ. For example, the NB15 has 2.5M samples in the *heterogeneous* scenario, and 1.6M in the *uniform* scenario. Moreover, let us focus on the performance achieved by past works. We can see that the state-of-the-art reaches very high F1-scores, which can raise the question of whether there is any point in improving such values. Nevertheless, we stress that cross-evaluations have a different objective (cf. Section 2.1.2): assessing the effectiveness of ML-NIDS against different attacks (*not included* in the respective datasets), and – if necessary – improving such ML-NIDS against these attacks. As our results will show, most of these ML-NIDS will perform poorly against different attacks, but can be strengthened; such achievements, however, could only be obtained by cross-evaluations.

TABLE 2.2: Statistics of the analyzed NID datasets.

| Scenario | Dataset | #Samples | #Attacks | #Features | F1-score |
|----------------|-----------|----------|----------|-----------|--------------|
| <i>Heter.</i> | CTU13 | 20.7M | 5 | 14 | 99.1% [11] |
| | NB15 | 2.5M | 9 | 48 | 98.7% [135] |
| | IDS18 | 3.1M | 14 | 80 | 96.2% [299] |
| | DDOS19 | 70M | 18 | 80 | 99.0% [234] |
| <i>Uniform</i> | UF-BotIoT | 600K | 4 | 12 | 97.0% [255] |
| | UF-NB15 | 1.6M | 9 | 12 | 85.0% [255] |
| | UF-IDS18 | 8.3M | 14 | 12 | 83.0% [255] |
| | UF-ToNIoT | 1.4M | 9 | 12 | 100.0% [255] |

Overall, these datasets contain traffic captured in large networks and the included malicious samples belong to a broad range of attacks¹⁴ Table 2.3 shows the attack distribution of the input ID for both scenarios. For simplicity, we organize Table 2.3 on the basis of three ‘families’ of attacks:

- *DoS*, for Denial of Service attacks (e.g., *DoS-Hulk*);
- *Botnet*, for Botnet attacks (e.g., *Rbot*);
- *Other*, for remaining attacks (e.g., *shellcode*, *scanning*).

¹³UF-NB15 and UF-IDS18 are generated from NB15 and IDS18.

¹⁴For a precise description of each attack, we refer the reader to the source material provided by the creators of each dataset.

We remark that, in our implementation of XeNIDS, we *always* use the specific attack classes, i.e., we do not ‘aggregate’ multiple attacks into a single class. The differentiation provided in Table 2.3 is for comprehensiveness, because the amount of specific attacks of our testbed is very broad. As an example, NB15 (and, hence, UF-NB15) has samples for all families, i.e., 2 different types of *Botnet* attacks, 1 type of *DoS*, and 6 types of *Other* attacks; whereas CTU13 only has samples for 5 different attacks of the *Botnet* family. From Table 2.3 we also determine that $\mu=46$ in the *heterogeneous* scenario, and that $\mu=36$ in the *uniform* scenario—this is because all the specific attack types are distinct across the input ID datasets.

TABLE 2.3: Distribution of attacks in each dataset. In our implementation of XeNIDS, we *always* use the specific attack classes, and perform no merging.

| <i>Heterogeneous scenario</i> | | | | <i>Uniform scenario</i> | | | |
|-------------------------------|---------------|------------|--------------|-------------------------|---------------|------------|--------------|
| Dataset | <i>Botnet</i> | <i>DoS</i> | <i>Other</i> | Dataset | <i>Botnet</i> | <i>DoS</i> | <i>Other</i> |
| CTU13 | 5 | 0 | 0 | UF-BotIoT | 0 | 2 | 2 |
| NB15 | 2 | 1 | 6 | UF-NB15 | 2 | 1 | 6 |
| IDS18 | 1 | 5 | 8 | UF-IDS18 | 1 | 5 | 8 |
| DDOS19 | 0 | 18 | 0 | UF-ToNIoT | 0 | 2 | 7 |
| Total | 8 | 24 | 14 | Total | 3 | 10 | 23 |

2.4.2 Preprocessing

We now describe the preprocessing computed on each considered NID dataset $D_i \in \mathbb{D}$ for both scenarios. Such operations represent the first two stages of XeNIDS: standardize (Section 2.3.2) and isolate (Section 2.3.3). Our low-level implementation of XeNIDS aims to overcome all the residual challenges in Section 2.2.3—to the extent this is possible with the current state-of-the-art. The experimental platform is an Ubuntu 20.04 machine with 64GB RAM and an Intel Xeon E5-2620 CPU. The development leverages the Scikit-Learn suite.

Standardize. We first associate each sample to its ground truth¹⁵. Then, we derive a common feature set based on the official NetFlow v9 documentation, which we report in Table 2.4. These features represent the minimum set of common features obtainable from the source data for both scenarios. We note that some datasets are provided with more features (e.g., IDS18 has 80), which are left out. However, as we will show in our experiments, the considered features yield ML-NIDS with state-of-the-art performance.

To sanitize network artifacts, we follow the recommendations in Appendix 2.8.2. To avoid overfitting and simulate the application of anonymisation techniques, we do not use the plain IP addresses or service-ports as features. Instead, we differentiate between internal/external hosts of each network (features 1 and 2 in Table 2.4); and we categorize the network ports according to the IANA guidelines (features 3 and 4 in Table 2.4). All of these operations are also adopted by recent works (e.g. [11]). We set the d of all samples in seconds, and we ensure that most samples fall within the same duration range (i.e., [0-150]s), discarding the few outliers.

Isolate. For each dataset D_i in \mathbb{D} , we separate benign from malicious samples using the ground truth label. We do not make any aggregation, hence our μ are the original ones (i.e., $\mu=46$ for the *heterogeneous* scenario, and $\mu=36$ for the *uniform* scenario). We thus obtain the following:

¹⁵We verify the checksum of each dataset, if provided.

TABLE 2.4: Feature set of our XeNIDS implementation in both scenarios.

| # | Feature Name | Type |
|----|---------------------------------|------|
| 1 | Source IP address internal | Bool |
| 2 | Destination IP address internal | Bool |
| 3 | Source port type | Cat |
| 4 | Destination port type | Cat |
| 5 | Flow Duration [s] | Num |
| 6 | Flow Direction | Bool |
| 7 | Incoming Bytes | Num |
| 8 | Outgoing Bytes | Num |
| 9 | Total Bytes | Num |
| 10 | Incoming Packets | Num |
| 11 | Outgoing Packets | Num |
| 12 | Total Packets | Num |

- for the *heterogeneous* scenario, \mathbb{N} containing 4 elements representing the source networks of the respective datasets (CTU13, NB15, IDS18, DDOS19), and \mathbb{M} containing 184 elements (because $n=4$ and $\mu=46$);
- for the *uniform* scenario, \mathbb{N} containing 4 elements representing the source networks of the respective datasets (UF-BotIoT, UF-NB15, UF-IDS18, UF-TonIoT), and \mathbb{M} containing 144 elements (because $n=4$ and $\mu=36$).

XeNIDS can now create the contexts to be cross-evaluated.

2.4.3 Assessment

In both scenarios we analyse three context types: C1, C4 and C7 (cf. Table 2.1). Let us explain the common assessment procedures, focusing on the architecture of the ML-NIDS and the considered performance metrics.

Parameters and Performance Metrics. We use the same parameters for our implementation of XeNIDS. Specifically, the adopted splits $s(N)$ and $s(M)$ are always 80:20 for both T and E . We use such splits because they are common in related literature (e.g., [11, 327]), therefore enabling a more fair comparison of our results with those of past works. We considered different ML algorithms, but we found that Random Forests consistently provided the best tradeoff in terms of detection performance, rate of false alarms, and training time—a result that confirms the state-of-the-art on the same datasets (e.g., [327, 11, 234, 255]). Hence our results will refer to Random Forest as the learning algorithm for each classifier. The performance metrics of interest are the *F1-score* (F1) and the *False Positive Rate* (FPR), defined as follows:

$$F1 = \frac{tp}{.5(fp+fn)+tp}, \quad FPR = \frac{fp}{tp+fn}, \quad (2.6)$$

where tp , fp , fn denote true positives, false positives, and false negatives, respectively; we consider a “true positive” as the correct detection of a malicious sample. It is desirable that the application of XeNIDS when considering modifications of the training set (hence, C7) should preserve the baseline FPR (cf. Section 2.2.3). Finally, to account for the randomness of each split, we repeat each experiment 5 times, and in our results, we will report the average of each repetition.

ML-NIDS Architecture. XeNIDS fosters development of *ensembles* of detectors (Section 2.3.4), each specialized in a single attack. There are many ways in which

such detectors can be integrated in a ML-NIDS. In our implementation, we assume that the NIDS uses ML as a final confirmation of detection. Hence, each sample is forwarded to the ‘most suitable’ detector of the ensemble, which must determine whether such sample is really malicious or not. While such selection is straightforward for contexts of type C1 and C7 (because $\bar{\tau} = \bar{\epsilon}$), this is not the case for C4, where the ML-NIDS is tested against ‘unknown’ attacks (because $\bar{\tau} \neq \bar{\epsilon}$). Hence, for C4 we perform a preliminary exploratory operation to identify the most suitable detector of the ensemble against the unknown attacks; this is to allow a more fair comparison with [234], which also investigates C4 by using two datasets considered in our testbed, IDS18 and DDOS19. Hence, we reserve a portion of the samples of each unknown malicious class, and test every detector composing the ML-NIDS against such portion: the one with the best performance is chosen as the candidate for analyzing the corresponding attack. This is legitimate because the ground truth of such samples is known, and such samples (i) are never used in E , and (ii) are not added in T (otherwise it would not be C4). Therefore, when presenting the corresponding results, we will report the performance achieved by the most optimal detector against each specific attack.

2.5 Demonstration

Our demonstration aims to simulate the exemplary use-cases described in Section 2.1.2. Let us discuss how we organize our demonstration by using the three considered types of contexts (C1, C4 and C7) enabled by the proposed model (cf. Table 2.1).

Workflow. We follow the same workflow for both the *uniform* and *heterogeneous* scenario.

1. Baseline (Section 2.5.1). We begin by assessing the case where the organization \mathcal{O} has an N and a M collected in their own network o . Such setup corresponds to context C1. To simulate C1, we use XeNIDS to devise a ML-NIDS for each dataset; such ML-NIDS is composed by an ensemble of detectors, each trained on a single attack contained in the same dataset. The ML-NIDS is tested against all the attacks of the ‘origin’ dataset. The expectation is that the results match the state-of-the-art.
2. Generalization Section 2.5.2. Having a ML-NIDS, the organization \mathcal{O} wants to assess its effectiveness against different attacks not included in T and originating from a different network than o . Such setup corresponds to context C4. We use XeNIDS to *test* the ‘baseline’ detectors of C1 against *all* attacks of *all* datasets. The expectation is that the performance will decrease substantially.
3. Extension (Section 2.5.3). To compensate for the low performance against unknown attacks, the organization \mathcal{O} borrows more malicious samples to improve the detection capabilities of their ML-NIDS. This corresponds to context C7 where \bar{t} extends \bar{o} . For each dataset, XeNIDS trains additional detectors by using the malicious samples of *all* the other datasets, and adds such detectors to the ensemble of the ‘baseline’ ML-NIDS. Such ‘extended’ ML-NIDS is tested against *all* attacks of the CS. The expected result is an improved performance w.r.t. C4.
4. Surrogation (Section 2.5.4). If the organization \mathcal{O} only has benign samples N from their own network o but does not have an M , the only option is using an

M from a different network to develop a ‘surrogate’ ML-NIDS. This is also represented by C7, but in this case \bar{o} and \bar{i} are disjointed. Hence, for each dataset, XeNIDS uses *only* the additional detectors developed at the previous step to devise a (new) ‘surrogate’ ML-NIDS. Such surrogate ML-NIDS is then tested only against the ‘attacks from different networks.

Example. Let us provide a complete example of the workflow above. We adopt the viewpoint of an organization that owns the UF-UNB15 network (hence, the *uniform* scenario). A total of 9 attacks originate from such network: 2 botnets, 1 DoS, and 6 others (cf. Table 2.3).

1. XeNIDS uses the 9 attacks of UF-UNB15 to train 9 detectors, representing the baseline ML-NIDS, which is tested against these 9 attacks.
2. XeNIDS tests the baseline ML-NIDS (with its 9 detectors) against all the attacks of all datasets. Namely: 4 attacks for UF-BotIoT, 14 attacks for UF-IDS18, 9 attacks for UF-ToNIoT, as well as the 9 in UF-UNB15.
3. XeNIDS trains 27 additional detectors, each using the benign samples of UF-UNB15 alongside the malicious samples of a specific attack contained in UF-IDS18, UF-ToNIoT, UF-BotIoT, respectively. Such detectors are *combined* with the 9 ‘baseline’ detectors of UF-UNB15, to extend the ML-NIDS. Such ‘extended’ ML-NIDS is tested against *all* the attacks of *all* datasets (36 attacks).
4. XeNIDS uses only the 27 detectors trained in the previous step (representing the ‘surrogate’ ML-NIDS) and tests them against the attacks contained in the corresponding networks, i.e., without taking into account the attacks (and the detectors) in UF-UNB15.

Such workflow is followed 4 times for both scenarios, each time by considering a different dataset as ‘origin’.

2.5.1 Baseline

We start by assessing C1, and report the results in Table 2.5. Specifically, on the left, we present the results for the *heterogeneous* scenario, and on the right, the *uniform* scenario. For each dataset, we report the average F1-score obtained against each family of attacks (cf. Table 2.3). Moreover, we report in the captions the average *FPR* achieved by the ML-NIDS of each dataset. Henceforth, all our results will be reported in the same format as Table 2.5.

TABLE 2.5: Baseline–C1. The table shows the F1-score of the ML-NIDS for each origin network. The performance matches the state-of-the-art. The FPR is less than 0.001 for all networks aside from UF-BotIoT (FPR = 0.11).

| <i>Heterogeneous scenario</i> | | | | <i>Uniform scenario</i> | | | |
|-------------------------------|---------------|------------|--------------|-------------------------|---------------|------------|--------------|
| Dataset | <i>Botnet</i> | <i>DoS</i> | <i>Other</i> | Dataset | <i>Botnet</i> | <i>DoS</i> | <i>Other</i> |
| CTU13 | 98.1 | — | — | UF-BotIoT | — | 99.9 | 92.0 |
| NB15 | 88.6 | 98.5 | 98.1 | UF-NB15 | 83.4 | 91.4 | 95.8 |
| IDS18 | 90.1 | 99.9 | 96.1 | UF-IDS18 | 99.9 | 99.1 | 99.2 |
| DDOS19 | — | 99.9 | — | UF-ToNIoT | — | 99.9 | 99.7 |

From Table 2.5, we observe that there are only two scores for UF-BoTIoT and UF-ToNIoT, because there are no *Botnet* samples in these ‘origin’ datasets. Similarly, CTU13 and DDOS19 presents only one score.

All our baseline detectors match the performance of past works (cf. Table 2.5). As an example, for the *uniform* scenario, the ‘worst’ ML-NIDS is trained (and evaluated) on UF-UNB15, but also in [255] such ML-NIDS achieves an average F1-score of 85%. Similarly, in the *heterogeneous* scenario, the ML-NIDS in [234] achieve 99.0 F1-score on both IDS18 and DDOS19, whereas [11] achieves 99.0 F1 on CTU13—all these results align with ours, confirming that our XeNIDS implementation is efficient.

2.5.2 Generalization

We then assess the baseline ML-NIDS when they are subject to attacks also contained in different networks, i.e., C4. We report the detection results in Table 2.6; the FPR is the same as in the baseline C1 (cf. caption of Table 2.5): this is expected because in C4 uses the same training sets as C1, and also the benign samples of the evaluation sets are the same as in C1,

From Table 2.6, we observe that the performance decreases because most of the attacks are ‘unknown’ to the baseline ML-NIDS. However, we can observe some interesting phenomena.

In the *uniform* scenario, the ML-NIDS of UF-ToNIoT can detect botnet attacks remarkably well (82% F1-score), despite *having no* ML-model specialized on botnet attacks (because no such attacks are contained in UF-ToNIoT). Such an intriguing finding could only be appreciated by cross-evaluating the ML-NIDS trained on UF-ToNIoT against malicious samples from different networks. Furthermore, the *heterogeneous* scenario shows that the baseline ML-NIDS of DDOS19 works very well against DoS attacks from other networks—despite such attacks being performed by different means.

We also compare some of our results with those in [234], which also investigated C4. Specifically, the ML-NIDS trained on DDOS19 and tested on IDS18 in [234] achieves 64% F1-score on average, which is similar to ours. Conversely, the ML-NIDS trained on IDS18 and tested on DDOS19 in [234] achieves an average 78% F1-score, which is slightly superior than ours. We explain this difference to the different conditions in [234]: they only consider a smaller portion of the initial dataset, whereas we use all of them. Hence, our samples may present a more skewed distribution that makes them more difficult to classify.

TABLE 2.6: Generalization–C4. Each baseline ML-NIDS of C1 is tested against the attacks of all other networks. Most attacks are not detected, and the F1-score degrades. The FPR is the same as in C1 because the benign samples are always the same and the training set is not modified.

| <i>Heterogeneous scenario</i> | | | | <i>Uniform scenario</i> | | | |
|-------------------------------|---------------|------------|--------------|-------------------------|---------------|------------|--------------|
| Dataset | <i>Botnet</i> | <i>DoS</i> | <i>Other</i> | Dataset | <i>Botnet</i> | <i>DoS</i> | <i>Other</i> |
| CTU13 | 80.0 | 38.1 | 49.7 | UF-BoTIoT | 47.8 | 69.0 | 76.8 |
| NB15 | 65.8 | 40.7 | 75.2 | UF-NB15 | 72.2 | 52.3 | 64.1 |
| IDS18 | 54.9 | 49.4 | 76.1 | UF-IDS18 | 68.2 | 81.0 | 63.3 |
| DDOS19 | 54.4 | 99.5 | 83.1 | UF-ToNIoT | 82.1 | 89.3 | 85.1 |

2.5.3 Extension

Next, we assess C7 when $\bar{\tau}$ extends $\bar{\delta}$, and report the results in Table 2.7. We observe that the overall performance increases (w.r.t. Table 2.6) by augmenting the training sets with the corresponding malicious samples. In the heterogeneous scenario, our ‘extended’ ML-NIDS naturally outperform those in [234], but we cannot claim this as a contribution because our ‘extended’ ML-NIDS use an augmented training set. We also appreciate that the FPR remains stable (cf. Table 2.5). We owe such results to our reliance on ensembles of detectors.

TABLE 2.7: Extension–C7. By augmenting the training set of the ML-NIDS with the malicious samples, the F1-score improves w.r.t. C4. The average FPR is lower than 0.001 for all networks aside from UF-BotIoT (FPR = 0.01).

| <i>Heterogeneous scenario</i> | | | | <i>Uniform scenario</i> | | | |
|-------------------------------|--------|------|-------|-------------------------|--------|------|-------|
| Dataset | Botnet | DoS | Other | Dataset | Botnet | DoS | Other |
| CTU13 | 98.8 | 99.9 | 98.9 | UF-BotIoT | 99.7 | 99.9 | 99.2 |
| NB15 | 97.1 | 99.9 | 99.1 | UF-NB15 | 88.9 | 99.2 | 98.7 |
| IDS18 | 98.5 | 99.7 | 97.7 | UF-IDS18 | 99.9 | 99.4 | 97.8 |
| DDOS19 | 99.9 | 99.9 | 98.6 | UF-ToNIoT | 99.7 | 99.9 | 99.9 |

2.5.4 Surrogation

Finally, we assess C7 when $\bar{\tau}$ and $\bar{\delta}$ are disjointed, and report the results in Table 2.8. From this table, we observe that all detectors exhibit very high F1-scores, implying that the malicious samples are considerably different than the benign samples. Sometimes, the F1-score reaches 99.9%, but is not perfect: we believe such occurrence to be *positive* because an F1-score of 100% could be related to overfitting.

TABLE 2.8: Surrogation–C7. We exclude all malicious samples (and detectors) from the each ‘origin’ network. The extremely high performance must be investigated. The average FPR is less than 0.001 and 0.0001 for all networks of *uniform* and *heterogeneous* scenarios, respectively.

| <i>Heterogeneous scenario</i> | | | | <i>Uniform scenario</i> | | | |
|-------------------------------|--------|------|-------|-------------------------|--------|------|-------|
| Dataset | Botnet | DoS | Other | Dataset | Botnet | DoS | Other |
| CTU13 | 99.9 | 99.9 | 98.9 | UF-BotIoT | 99.7 | 99.9 | 99.9 |
| NB15 | 99.9 | 99.9 | 99.9 | UF-NB15 | 99.9 | 99.9 | 99.9 |
| IDS18 | 99.6 | 99.6 | 99.8 | UF-IDS18 | 99.9 | 99.9 | 99.9 |
| DDOS19 | 99.9 | 99.9 | 98.6 | UF-ToNIoT | 99.7 | 99.9 | 99.9 |

2.6 Discussion

We now discuss the results presented in Section 2.5. We first summarize the main findings (Section 2.6.1), and then make some considerations *reliability* of the results (Section 2.6.2 and Section 2.6.3). We then present the main limitations of our demonstration, as well as possible workarounds (Section 2.6.4).

2.6.1 Preliminary Analysis

We appreciate that, in general, our results show the effectiveness of XeNIDS in producing baselines with state-of-the-art performance, while also extending the detection surface. It is intriguing that, in some cases, it is possible to detect attacks without training on the related malicious samples. To further stress the advantages of cross-evaluations, we provide an in-depth look at our results by focusing on the CTU13 dataset. This dataset contains *only botnet* attacks and, from Table 2.2, the state-of-the-art (e.g., [11]) achieves 99.1% F1-score against such attacks. A similar performance may suggest that improvements can be incremental at best; however, no past works have assessed how ML-NIDS trained on CTU13 can detect different *botnet* attacks (not included in CTU13). By applying the proposed XeNIDS framework, we discover that similar ML-NIDS perform much worse: as shown by Table 2.6 (Section 2.5.2), the F1-score of such ML-NIDS drops by 20% against *botnet* attacks of diverse datasets; even worse, it is unable to detect *DoS* attacks (F1-score of 38%). *Such poor performance could only be assessed via cross-evaluations.* To make it better, the performance against these – different – attacks can be increased by training on the respective samples: by observing Table 2.7, the F1-score can be restored to 99% via cross-evaluations. Also noteworthy is that the FPR always remains within acceptable levels (below 0.001). Such FPR will resemble the one after deployment (because the source of benign samples is always the same).

However, as stated in Section 2.2.3, it is necessary to further analyze the results of XeNIDS. This is to avoid relying on a false-sense of security, given by high performance at test-time which does not correspond to the performance after the ML-NIDS is deployed. We specifically focus on contexts of type C7 because they involve modifications of the training data, which can lead to ‘network artifacts’ that affects the *Env* component of NetFlows (cf. Expression 2.5 in Section 2.3.2) and, potentially, lead to overfitted ML-NIDS.

2.6.2 Reliability: Uniform scenario

In this scenario, by definition, the *Env* is affected only by *NetId* because *Conf* is the same for all datasets; such characteristic implicitly reduces the risk of network artifacts. Nevertheless, we find instructive to analyze the results of the surrogate ML-NIDS, reported in the right-side of Table 2.8. In particular, we consider the UF-UNB15 network. We observe that the ‘surrogate’ detectors focused on botnet attacks achieve a near-perfect F1-score, which is *higher* than both their ‘extended’ and ‘baseline’ variants (cf. Tables 2.7 and 2.5). This implies that benign samples of UF-UNB15 are very similar to the (malicious) botnet samples of UF-UNB15, making such botnet samples harder to classify by the UF-UNB15 ML-NIDS w.r.t. the botnet samples in other networks. Such occurrence can be a sign of overfitting, because the UF-UNB15 ML-NIDS could be detecting the botnet samples from other networks on the basis of network artifacts. However, a more detailed analysis can remove such doubt. Indeed, in the uniform scenario, the only other source of ‘botnet’ samples is UF-IDS18, where the baseline performance is also perfect (cf. Table 2.5), a result also confirmed by the state-of-the-art [255]. Simply put, the ‘botnet’ samples in UF-IDS18 are easy to identify. Such observation reduces the chance that the surrogate (or the extended) ML-NIDS of UF-UNB15 are affected by artifacts from UF-IDS18.

2.6.3 Reliability: Heterogeneous scenario

This scenario assumes NetFlows generated via different means, hence the *Env* component is affected by both *NetId* and *Conf*. Such characteristic increases the chance that some artifacts ‘evaded’ XeNIDS standardize stage. To find a trace of such artifacts, we compare the *feature importances* of each ML-NIDS (all ML-NIDS use the same feature set).

Intuitively, the most important features for detecting an attack in its ‘origin’ network should denote the malicious behavior—hence, such features should be also the most important when the attack is ‘transferred’ to train a different ML-NIDS (which is the case in C7). We provide in Figure 2.8 a comparison of such importances, focusing on the detectors specialized on the *Rbot* botnet attack (contained in the CTU13 network). Specifically, Figure 2.8 shows the importances of the top6 most important features (out of 12—cf. Table 2.4) for all the *Rbot* detectors among the four different networks.

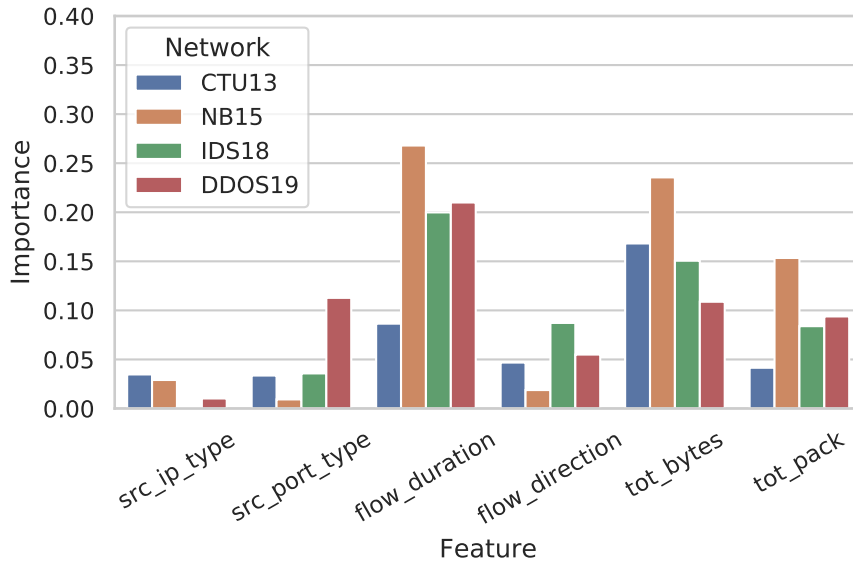


FIGURE 2.8: Feature importances of the *Rbot* detectors (Heterogeneous CS).

From Figure 2.8 we observe that the detectors can either ‘agree’ or ‘disagree’ on the importance of such features. Specifically, we observe that the ‘origin’ CTU13 detector (blue bars) places a great importance on the *tot_bytes*, denoting agreement with the other detectors; however, there is disagreement on the *duration*, which is less important for the CTU13 detector. The general trend in Figure 2.8 is that the detectors disagree on most features: therefore, we cannot exclude that some underlying effects of *Env* are still present.

2.6.4 Limitations and Future Work

To increase the reliability of the detection performance, it is necessary to assume the perspective of the *owners* of each network. As a practical example that could remove any doubt, the owners of the NB15 network should infect their machines with the *Rbot* botnet (contained in CTU13), and verify whether their ML-NIDS (trained on the *Rbot* samples from CTU13) can detect such attack. Doing such verifications is not

possible for our scientific paper, as they require a complete control and overview of the monitored network. Moreover, the CnC servers of the *Rbot* botnet are no longer active. Our experiments are for demonstrative purposes, but realistic deployments should integrate such verifications—which *must* be done regardless of the origin of the malicious samples (i.e., both in ‘traditional’ and in ‘cross’ evaluations).

Moreover, we note that each considered context type is an independent use case. Indeed, our focus is not on developing systems that outperform the state-of-the-art: it would be unfair to claim that our ML-NIDS generated via C7 are better than those in [234]. In contrast, our goal is to demonstrate the contexts that can be assessed by mixing different network datasets, showcasing the potential of such cross-evaluation for the state-of-the-art. As such, we can consider our results as a ‘benchmark’, allowing future cross-evaluation studies to compare their results with those in our paper.

Finally, an intriguing future research direction is the assessment of cross-evaluations in adversarial settings: for instance, how would a ML-NIDS poisoned with samples from a different network perform (cf. Section 2.2.3)? Answering a similar question would be beneficial to the ML-NIDS research area.

2.7 Summary

Despite many successes, the integration of supervised Machine Learning (ML) methods in Network Intrusion Detection Systems (NIDS) is still at an early stage. This is due to the difficulty in obtaining comprehensive sets of labelled data for training and evaluating a ML-NIDS. The recent release of labelled datasets for ML-NIDS was appreciated by the research community; however, few works noticed the opportunity that such availability provides to the state-of-the-art.

Inspired by the necessity of proactive empirical evaluations and the recent release of more open datasets, we promote the idea of cross-evaluating ML-NIDS by using existing labelled data from different networks. Such approach has been applied before, but no past work specifically tackled this problem. As a result, all the benefits of cross-evaluations, as well as their intrinsic risks, are still unexplored.

We address all of these issues in this paper. We begin by presenting the first model for cross-evaluation of ML-NIDS, which is data-agnostic and general enough to cover both supervised and unsupervised ML-NIDS. By using such model, we highlight the limited scope adopted by most related works, and showcase the benefits provided by cross-evaluations of ML-NIDS. We also present all the challenges and limitations of such opportunity, which must be known and adequately addressed in order to provide actionable results.

To foster proactive cross-evaluations, we develop XeNIDS, the first framework for cross-evaluations of ML-NIDS. XeNIDS aims to mitigate all the hazards arising from using data from different networks. Specifically, XeNIDS focuses on NetFlow data, which is popular in the ML-NIDS community due to its flexibility and suitability for detection purposes.

Finally, we elucidate the potential of cross-evaluations via a large set of experiments, where we use XeNIDS to cross-evaluate ML-NIDS on 6 well-known datasets. In our demonstration, we show the capability of XeNIDS to retain the ‘baseline’ performance of past ML-NIDS, while illustrating some additional use-cases enabled by cross-evaluations, such as ‘extending’ the detection surface of ML-NIDS. We conclude our demonstration with a follow-up discussion where we question the reliability of the results, which is necessary for realistic deployments of ML-NIDS.

Our paper will hopefully inspire future works on ML-NIDS, and is oriented to both researchers and practitioners. The former can make better use of open datasets to cross-evaluate past and future ML-NIDS, allowing broader assessments of the state-of-the-art; the latter can use future research results, or completely integrate cross-evaluations in their proactive assessments, to develop or improve Machine Learning-based Network Intrusion Detection Systems—without incurring in extra labelling procedures. We believe that cross-evaluations – supported by data-sharing platforms and federated learning techniques – represent a pragmatic way to overcome the specificity of NIDS and realize ‘general’ ML-NIDS.

2.8 Appendix

2.8.1 Contributors to NetFlows

Let us illustrate the role played by *Comm* and *Env* (i.e., *NetId* and *Conf*) in the generation of the corresponding NetFlows (see Expression 2.5). Assume that two organizations, \mathcal{O}_1 and \mathcal{O}_2 , have two distinct networks both having a pair of hosts (h_1^1 and h_2^1 for \mathcal{O}_1 , h_1^2 and h_2^2 for \mathcal{O}_2); such hosts communicate with each other within their own networks. It is straightforward that if these two pairs of hosts exchange different information (viz., resulting in different *Comm*) then the resulting NetFlows generated in \mathcal{O}_1 and \mathcal{O}_2 will differ. Let us focus on the case where the pairs of hosts exchange the same information (viz., same *Comm*). For simplicity, assume that the first host of each pair (h^1) sends *exactly* the same file of 100MB to the second host, using *exactly* the same protocol and ports. Let us assume that the hosts in \mathcal{O}_1 are allocated a bandwidth of b_1 Mb/s, and that those in \mathcal{O}_2 are allocated a bandwidth of b_2 Mb/s. Finally, let us assume that the organizations use the same NetFlow generation software, configured to allow the maximum duration of a NetFlow to be d_1 for \mathcal{O}_1 , and d_2 for \mathcal{O}_2 . We identify four scenarios.

- $b_1 = b_2$ and $d_1 = d_2 \rightarrow$ same *NetId* and same *Conf* (viz. same *Env*). For instance, if $b_1 = b_2 = 100\text{Mb/s}$ and $d_1 = d_2 = 10\text{s}$, then the file will be transferred in the same amount of time (8s) in both \mathcal{O}_1 and \mathcal{O}_2 , resulting in similar NetFlows (with a duration of 8s).
- $b_1 \neq b_2$ and $d_1 = d_2 \rightarrow$ different *NetId* but same *Conf* (viz. different *Env*). For instance, if $b_1 = 100\text{Mb/s}$ and $b_2 = 1\text{Mb/s}$, then the file will be transferred in 8s in \mathcal{O}_1 but in 800s in \mathcal{O}_2 , resulting in different NetFlows.
- $b_1 = b_2$ and $d_1 \neq d_2 \rightarrow$ same *NetId* but different *Conf* (viz. different *Env*). For instance, if $b_1 = b_2 = 100\text{Mb/s}$ while $d_1 = 10\text{s}$ and $d_2 = 1\text{s}$, then the transfer will take 8s in both \mathcal{O}_1 and \mathcal{O}_2 ; but in \mathcal{O}_1 there will be 1 NetFlow of 8s, while in \mathcal{O}_2 there will be 8 NetFlows of 1s.
- $b_1 \neq b_2$ and $d_1 \neq d_2 \rightarrow$ different *NetId* and different *Conf* (viz. different *Env*). This is self-explanatory.

Of course, there are many other factors that affect *NetId* and *Conf* (aside from the bandwidth and maximum duration). The above-mentioned example is just for demonstrative purposes.

2.8.2 Guidelines for Standardize

To avoid generating network specific artifacts (discussed in Section 2.3.2), we provide some recommendations on three common NetFlow fields: the *IP addresses*, the *service ports*, and the *duration*.

IP addresses. There are two issues that may arise when standardizing the IP addresses of two distinct datasets:

- different networks use different subnet masks. For instance, the internal IP addresses of D_i may present the structure “192.168.x.x”, whereas those in D_j are “175.32.x.x”;
- the malicious traffic of a given dataset may be entirely produced by just few machines.

Neglecting these issues may result in ML models that distinguish legitimate from anomalous samples on the sole basis of the IP address of a host, without giving the due importance to the remaining traffic characteristic. This is a problem because if a *real* attack involves a machine with a different IP address, the detector would never identify it. We hence propose to standardize each dataset by separating *internal* from *external* hosts. The ML model will use these features, instead of the IP addresses, to perform its analyses. The information to perform this separation can be obtained either from the documentation of a dataset, or by inferring it from the data using expert knowledge; if such information is not obtainable, then we suggest not to use any IP-related feature.

Service Ports. Handling the service ports of distinct datasets presents similar issues to the IP addresses discussed above: different networks may adopt different port policies; and the attacks captured by a given dataset may rely just on a restricted (or unique) set of ports. We thus propose to standardize each dataset by categorizing each port on the basis of the IANA guidelines, i.e., *well-known* [0-1023], *registered* [1024-49151] and *dynamic* [49151-65535].

Duration. Besides verifying that all datasets use the same measurement units, standardizing the NetFlow duration (d) of distinct datasets is a challenging task. On the one hand, datasets may be created with different NetFlow tools and/or different configuration parameters. For example, setting the maximum duration (d_{max}) of a NetFlow to 1000 or 100 seconds would lead to significantly different results¹⁶. On the other hand, there may be some underlying traits of a given network that lead its machines to generate flows of different duration. To address these issues, we propose three possible solutions, all involving the identification of the smallest maximum duration across all datasets, $\min(d_{max})$:

- *Outlier removal.* This approach assumes that (i) the duration of the majority of samples (from all considered datasets) falls within a reduced range $[d_l, d_t]$, and that (ii) the top limit d_t of this range is *lower* than $\min(d_{max})$. In these circumstances, it is possible to remove the few “outliers” that have extremely high durations with respect to the remaining samples. Despite the consequential loss of samples, removing outliers does not necessarily reduce the prediction performance.
- *Threshold setting.* This solution avoids data loss problems. If a dataset D_i has $d_{max} \gg \min(d_{max})$, its samples having $d > \min(d_{max})$ will have their duration

¹⁶This issue can be overcome if the datasets are provided in PCAP format by properly setting the NetFlow generation tool.

set to $\min(d_{max})$. However, it is important to store the original value of d if it is needed to compute some derived metrics, such as the *packets per second*. This approach may be unpractical for ML leveraging sequential analyses as it disrupts the sequence of samples.

- *Flow splitting*. This technique enables the application of sequential ML methods. The intuition is to *split* those flows that exceed $\min(d_{max})$. Given a D_i with $d_{max} > \min(d_{max})$, the idea is to truncate all flows of D_i with duration $d > \min(d_{max})$ into multiple flows. As a practical example assuming duration expressed in seconds, if $\min(d_{max})=300$ and D_i has $d_{max}=1000$, and if a given flow in D_i has $d=700$, the approach truncates this flow in three distinct flows, with $d=(300, 300, 100)$. When performing the split, it is important to also update some metrics, such as the transferred bytes or packets (which can be adjusted proportionally) as well as the start and finishing times of the flow.

We observe that, in our experiments, we adopt the *outlier removal* strategy. Despite being lossy, such technique still allows to devise ML-NIDS with performance matching the state-of-the-art (see Table 2.5 and compare it with Table 2.2).

2.8.3 Symbol Table

To facilitate the readability, we report in Table 2.9 the major notation used throughout the main sections of our paper.

We also further explain the difference between some of our symbols introduced in Section 2.2, and specifically the difference between the arrays and sets (e.g., \vec{t} and \bar{t}). Let us assume a scenario where $n=3$ and $\mu=3$, meaning that \mathbb{M} is a 3×3 matrix. We use the *ordered arrays* \vec{t} , $\vec{\tau}$ (or \vec{v} , \vec{e}) to answer the question “which elements of \mathbb{M} are included in T (or E)?”. A possibility is that $\vec{t}=(1,1,2)$ and that $\vec{\tau}=(2,3,3)$. This means that T will contain M_1^2 , M_1^3 , M_2^3 . Hence, $\bar{t}=(1,2)$ because \bar{t} is the *set* denoting the (unique) ‘malicious’ networks included in T . At the same time, $\bar{\tau}=(2,3)$ because $\bar{\tau}$ is the *set* denoting the (unique) attacks included in T .

Finally, we stress that, in our cross-evaluation model, $\bar{o}=\vec{o}=o$, because the origin of the benign samples must be the same for both the training and evaluation partitions (i.e., T and E , respectively).

TABLE 2.9: Table of relevant notation used in this paper.

| SYMBOL | DESCRIPTION | REF |
|--|---|---------------|
| \mathcal{O} | An organization | Section 2.1.2 |
| o | The network of the organization \mathcal{O} | Section 2.1.2 |
| N | A set of benign network samples | Section 2.1.2 |
| M | A set of malicious network samples | Section 2.1.2 |
| \mathbb{D} | A collection of datasets, each generated in a specific network | Section 2.2.1 |
| n | The cardinality of \mathbb{D} , i.e., the number of different networks included in \mathbb{D} | Section 2.2.1 |
| D_i | The set of samples generated by network i ($\bigcup_i D_i = \mathbb{D}$) | Section 2.2.1 |
| N_i, M_i | The set of benign and malicious samples included in D_i ($N_i \cup M_i = D_i$) | Section 2.2.1 |
| μ | The number of all attacks included in \mathbb{D} | Section 2.2.1 |
| M_i^α | The samples of network i corresponding to the attack α ($\bigcup_\alpha M_i^\alpha = M_i$) | Section 2.2.1 |
| \mathbb{N}, \mathbb{M} | The collection of all N and M included in \mathbb{D} | Section 2.2.1 |
| T, E | The training and evaluation sets of a ML-NIDS | Section 2.2.2 |
| N_o | The benign samples (from the same network o) used in both T and E | Section 2.2.2 |
| M_i^T | An element of \mathbb{M} used in T | Section 2.2.2 |
| M_e^E | An element of \mathbb{M} used in E | Section 2.2.2 |
| i, ε | The set of all networks included in T and E | Section 2.2.2 |
| $\bar{\varepsilon}, \bar{\varepsilon}$ | The set of all attacks included in T and E | Section 2.2.2 |
| $T(\delta, \bar{i}, \bar{\varepsilon})$ | The function describing the training set T | Section 2.2.2 |
| $E(\delta, \bar{\varepsilon}, \bar{\varepsilon})$ | The function describing the evaluation set E | Section 2.2.2 |
| $c(\delta, \bar{i}, \bar{\varepsilon}, \tau, \bar{\varepsilon})$ | A context is defined by the relationships between $\delta, \bar{i}, \tau, \bar{\varepsilon}, \bar{\varepsilon}$ | Section 2.2.2 |
| $Comm$ | The contribution to a NetFlow of the communications of the involved hosts | Section 2.3.2 |
| Env | The contribution to a NetFlow of the network environment of its two hosts | Section 2.3.2 |
| $NetId$ | The intrinsic properties of a network influencing Env | Section 2.3.2 |
| $Conf$ | The configuration of the NetFlow appliance influencing Env | Section 2.3.2 |
| \mathbb{T}, \mathbb{E} | The set of all T and E generated by XeNIDS | Section 2.3.4 |

Chapter 3

DETONAR: Detection of Routing Attacks in RPL-based IoT

The revolution of the internet world is happening in recent years, bringing the name of the Internet of Things (IoT). Initially studied as the evolution of Wireless Sensors Networks (WSNs) [194], recently IoT has gained much popularity and scientific community attention. The new paradigm introduced by IoT has shown the broadest range of applications, from industrial scenarios [36, 69], to smart homes [95, 275], intelligent healthcare [139], and smart cities [323].

The importance of Internet of Things (IoT) applications introduces the need to secure IoT networks [285, 150]. Indeed, many examples of attacks and vulnerabilities can be found for these networks. For instance, in October 2016, the largest Distributed-Denial-of-Service (DDoS) attack was launched using an IoT botnet. This specific attack leveraged the Mirai malware [158]. In 2017, a study developed by the US Food and Drug Administration confirmed that some cardiac devices present serious vulnerabilities that, if exploited, could allow unauthorized access to the devices [61].

There exists a broad variety of communication protocols commonly used in IoT networks, e.g., WiFi [171], IEEE 802.15.4 [74], RFID [141], Bluetooth [62]. Depending on the considered devices, the surrounding environment, and the required communication range, different protocols are leveraged. In this paper, we consider the standardized Routing Protocol for Low Power Lossy Networks (RPL). RPL is considered the *de facto* routing protocol for IoT and can be efficiently used in different applications, including but not limited to healthcare, smart environments, transport, industry, and military applications [151, 100, 155]. RPL is a proactive protocol developed to allow communication in wireless networks with low power consumption and generally susceptible to packet loss. In particular, we inspect the identified vulnerabilities of RPL. In recent years, research efforts shed light upon routing attacks against RPL (e.g. rank attack, version attack, etc.) [304, 233, 241, 64].

Given the wide variety of attacks available against this protocol, its secure deployment is difficult. Moreover, RPL's popularity in IoT applications renders the security problem of this protocol of paramount importance. In the recent past, some works have been proposed aiming at securing RPL with alternate success [232, 282, 99, 261, 6, 147, 65]. Although showing acceptable detection performances, state-of-the-art security systems focus only on few attacks while introducing communication overhead and computation performance issues. Therefore, there is a lack of a reliable and comprehensive IDS that can identify more attacks with low overhead. To this end, we propose DETONAR¹, an Intrusion Detection System used to DETect rOutiNg Attacks in RPL. DETONAR leverages traffic analysis technique.

¹DETONAR is a Venetian dialectal word that stands for detonating a bomb, as the proposed IDS aims at detonating any possible attacks against RPL.

Only a few works explore this concept in RPL security, such as [319, 34]. However, these works focus only on the identification of a few attacks. The proposed IDS aims to identify the maximum number of attacks while introducing zero RPL overhead and zero computations at IoT devices. DETONAR leverages traffic sniffer devices and traffic analysis techniques in order to build a centralized IDS. The DETONAR detection scheme's centrality allows it to overcome state-of-the-art drawbacks like RPL overhead and computational power requirements of IoT devices. Moreover, DETONAR introduces a hybrid approach relying on the combination of anomaly-based detection and signature-based classification techniques. Combining these techniques allows DETONAR to reach reliable attack detection while maintaining low false positives.

Given the unique nature of DETONAR and its use of network traffic analysis techniques, we need to introduce a novel dataset of attacks against RPL, called RADAR, to evaluate its performances. We consider 14 well-known routing attacks against RPL and use the NetSim tool [287] to implement these attacks and extract the corresponding dataset (see Section 3.2). The dataset consists of the packet trace files for each simulation. These files contain the packets that each device has sent during the communication period. The simulation implementation considers only static scenarios, given their popularity among RPL's applications and medium-sized networks (i.e. 16 nodes). To the best of our knowledge RADAR represents the biggest and most comprehensive dataset of routing attacks in RPL. Its introduction represents a step forward in security research as it allows to study a wide variety of attacks simultaneously.

Contribution To summarize, the contributions that our work brings to IoT security are the following:

- We present *RADAR*, a novel Routing Attacks Dataset for RPL. To the best of our knowledge, RADAR represents the first dataset containing RPL traffic for a vast range of attacks. The dataset (see Section 3.2 for more details) contains network traffic of simulations for 14 well known attacks. Thanks to NetSim, we make the dataset publicly available². RADAR contains 80 different simulations, each of length 1500 seconds, obtaining on average more than a million packets for each simulation.
- We present *DETONAR*, a novel IDS developed to detect routing attacks in RPL and identify intruders. Due to its novel sniffing approach and the centralized computation paradigm, DETONAR (see Section 3.3 for more details) maintains zero communication overhead at RPL level and requires no device computations or firmware update.
- We show the effectiveness of DETONAR on the proposed RADAR dataset and shows its applicability to small-scale networks. DETONAR's true positive detection exceeds 80% for 10 attacks out of 14 (see Section 3.4.2 for more details), while requiring relatively small computation time, due to its hybrid approach.

Organization This chapter is organized as follows. Section 3.1 discusses the basic concepts of RPL, also presenting the known attacks against it in IoT networks. Section 3.2 presents RADAR in details, explaining its most relevant features. Section 3.3 describes the proposed IDS, its workflow, and the main advantages

²<https://spritz.math.unipd.it/projects/detonar/>

that it brings. Section 3.4 presents the attack detection parameters optimization process and the performance analysis of our proposed IDS over different IoT scenarios. Finally, Section 3.5 provides conclusions and insight of possible extensions of our work.

3.1 Background and Related Work

In this section we briefly introduce the routing protocol for Low Power Lossy Networks (LLNs), called Routing Protocol for Low Power Lossy Networks (RPL) (see Section 3.1.1). Section 3.1.2 then summarizes well known networking attacks against RPL and their workflow. Finally, in Section 3.1.3 we present the available IDSs and their limitations, motivating our work.

3.1.1 Routing Protocol for LLNs (RPL)

RPL is the routing protocol standard for LLNs. This protocol was developed by the IETF ROLL task force and is specified in the Request For Comment (RFC) 6550 [308]. The development of RPL was commissioned by IEEE to bridge the existing gap in routing for IoT scenarios. The RPL protocol has been specifically designed to meet the requirements of resource, power, and bandwidth constrained devices.

The fundamental concept standing at the base of RPL is the topological notion of *Destination Oriented Directed Acyclic Graphs* (DODAGs). The DODAG is a directed graph-oriented towards a root node without loops. The nodes providing access to the internet (gateways) are considered root nodes. All other nodes link to them directly or through a series of parent nodes. Each node selects a preferred parent that is used to forward application packets. It is selected depending on the *rank* value that can be obtained by a device. This value represents the position of a node in the DODAG. *Rank* depends on both, the distance of the node from the root, and the *objective function* (OF). It describes how distance and signal-to-noise ratio are used to compute rank.

RPL introduces new control packets leveraged to build and maintain the DODAG and the communication routes. RPL control packets are defined as a new type of Internet Control Messages Protocol version 6 (ICMPv6) control packets. In particular, *DODAG Information Solicitation* (DIS) packets are used by a node to ask for information regarding neighbourhood status, and *DODAG Information Object* (DIO) packets are used to discover RPL instances, learn DODAG configurations, select preferred parent, and maintain DODAG structure. *Destination Advertisement Object* (DAO) packets are used to advertise reverse routes information, creating upward and downward paths between any nodes, and the *Destination Advertisement Object* (DAO-ACK) packets are sent in response to a DAO packet.

RPL supports different communication paradigms, that includes Point-to-multipoint (P2MP), point-to-point (P2P) and multipoint-to-point (MP2P). It also provides two *modes of operation*. In *storing* mode, the preferred parent will store routing information in a routing table. Using this mode, application packets reach the closest common parent before being redirected to destination. In *non-storing* mode, the root node is the only device that maintains a routing table. Using this mode, application packets reach root node before being redirected to destination. Full explanation of RPL implementation details is out of the scope of this paper. Interested readers may find more comprehensive literature on RPL protocol in [308, 154].

3.1.2 Attacks on RPL

There exist many networking attacks against RPL. Even if traditional security protocols are implemented (IPSec [244], SSL [145], etc.), RPL does not guarantee communication and routing security. Malicious user can take possession of a device, modify the DODAG structure, or block application packets. To study networking attacks against RPL, we refer to [304, 233, 241]. To the best of our knowledge, there exist 16 well-known attacks, each of them presenting specific behaviours.

- *Blackhole and Selective Forward attacks.* The attacker may drop all (*blackhole*) or some (*selective forward*) application packets received from its children. *Goal:* Denial-of-Service.
- *Sinkhole, Rank, and Continuous Sinkhole attacks.* A malicious user can fake its rank value, modifying or disrupting network structure. *Goal:* DODAG modification.
- *HELLO flooding, and DIS attacks.* A malicious device forges and sends a high amount of control packets (DIO or DIS). Forged messages keep the neighbours busy trying to process them. *Goal:* Network flooding.
- *Clone ID and Sybil attacks.* An attacker can advise himself as one or multiple different devices, stealing the identity of a legitimate node. *Goal:* Eavesdropping.
- *Wormhole attack.* Multiple attackers can collaborate to create a tunnel. The created tunnel allows the two malicious nodes to intercept and divert many applications packets. *Goal:* Routes modification.
- *Version, Local repair, DODAG inconsistency and Storing mode attacks.* A malicious user can forge modified control packets containing anomalous parameters. These packets provide to disrupt communication. *Goal:* DODAG disruption.
- *Replay attack.* An attacker can replay old control packets received from other devices. Settings inconsistencies make neighbours unable to communicate. *Goal:* DODAG disruption.
- *Worst Parent attack.* An attacker can select a new parent without changing its rank. The new parent is chosen to be the worst possible, creating sub-optimal paths. *Goal:* DODAG modification.
- *DODAG Inconsistency attack.* An attacker can misuse RPL's DODAG repair mechanism to attack the network. Manipulation of few packets' flags can trigger a DODAG repair mechanism, making it impossible for devices to communicate properly. *Goal:* DODAG disruption.
- *Storing Mode attack.* This attack requires RPL to run in storing mode. An attacker can advise many non existing routes to a legitimate device. The advised routes saturate the routing table of the compromised device, preventing it from building correct routes. *Goal:* Routes disruption.

Table 3.1 shows the characteristics of each well known routing attack against RPL. In particular, we consider if a strategy is influencing or disrupting the DODAG structure. We also study if an attack behaviour increases the end-to-end delay due to long queues created at each device or by producing sub-optimal routes. Packet reception may also be influenced due to high packet drop rate or communication overhead. Finally, we also consider if collaboration between attackers, or packet forgery are required for a specific attack.

TABLE 3.1: Main features of attacks against RPL. These features help to understand attacks behaviour and to classify them.

| Attack | DODAG | Queueing Delay | Routing Delay | Packet Loss Directly | Packet Loss via Overhead | Collaboration | Forgery |
|---------------------|-------|----------------|---------------|----------------------|--------------------------|---------------|---------|
| Blackhole | | | | ✓ | | | |
| Selective forward | | | | ✓ | | | |
| Sinkhole | ✓ | | ✓ | | | | |
| Continuous Sinkhole | ✓ | | ✓ | | | | |
| HELLO flooding | | ✓ | | | ✓ | | ✓ |
| Clone ID | | | ✓ | ✓ | | | |
| Sybil | | | ✓ | ✓ | | | |
| Wormhole | | | ✓ | | | ✓ | |
| Version | ✓ | | ✓ | | ✓ | | ✓ |
| Replay | ✓ | ✓ | ✓ | | ✓ | | ✓ |
| Rank | ✓ | | ✓ | | | | |
| Worst parent | ✓ | | ✓ | | | | |
| DIS | | ✓ | | | ✓ | | ✓ |
| Local repair | ✓ | | ✓ | | ✓ | | ✓ |
| DODAG inconsistency | | | ✓ | | ✓ | | ✓ |
| Storing mode | | ✓ | ✓ | | ✓ | | ✓ |

3.1.3 State-of-the-art Intrusion Detection Systems

Knowing the importance of security issues in RPL, many systems have been proposed to patch this protocol. Intrusion Detection Systems (IDSs) are the most popular mechanisms to detect security threats in a network [7, 177, 124]. These are used to find any possible intruder in the IoT network. In particular, *anomaly based* IDS aims at identifying non-legitimate behaviour, knowing how the network works when no attacker is present. These systems can work properly having the knowledge of legitimate network traffic only. On the other hand, any fluctuation from the legitimate behaviour is considered an anomaly. Therefore, anomaly based systems are characterized by high false positive rate. *Signature-based* IDS instead utilizes signatures of attacks behaviour to identify intruders. These systems are capable of obtaining low false positive rate, but are not flexible. Signatures are found for specific attack patterns, requiring their full knowledge.

Raza et al. [245] proposed a system based on report packets, which contain network information and are sent by IoT devices to the root node upon request, to secure RPL. Reports obtained by root node are then used to reconstruct the DODAG and find anomalies in its structure. DODAG and network information are used to detect sinkhole, blackhole, and selective forward attacks. In [232], the authors propose a system based on device location knowledge to identify wormhole attacks. Cervantes et al. [46] propose a modification of the RPL protocol to detect sinkhole attacks by using an IDS mechanism which is based on network clusterization. An extension of [46] was proposed by Surendar et al. [282]. Authors aim decrease overhead and increase packet delivery ratio. Gara et al. [99] focuses on mobile Wireless Sensor Networks (WSNs), in which they try to identify possible selective forward attacks. Authors in [261] propose an IDS that is based on game-theory strategies. The particular approach aims to decrease false positive rate, energy consumption, and overhead. A trust-based security mechanism was presented by Airehrour et al. [6] that was tested against rank and sybil attacks. Authors in [319] proposed a deep learning framework based on traffic analysis to detect rank, hello flooding and version attacks. Finally, Mayzaud et al. [199] proposed a distributed security mechanism based on monitoring devices, capable of sniffing network traffic, to detect version attacks.

The state-of-the-art systems show different possibilities to secure RPL based networks. Although showing some advantages, these mechanisms also present drawbacks, which are as follows.

1. *Scarn variety of attacks.* Existing IDSs are able to detect few class of attacks

3.2 RADAR: Routing Attacks Dataset for RPL

In this section, we present RADAR, a novel Routing Attacks DATaset for RPL. To the best of our knowledge, there exist no exhaustive datasets containing RPL traffic for a vast range of routing attacks. We aim at filling this lack proposing RADAR. This dataset represents a novelty both in terms of variety of considered routing attacks and extracted traffic. We use Netsim [287] to implement five simulations for 14 of the attacks presented in Section 3.1.2. Only *DODAG inconsistency* and *storing mode* attacks are not implemented due to NetSim’s software limitations. NetSim does not implement specific RPL flags that are required for these two attacks to be simulated. These flags have been introduced in later developments of RPL and are not required for its proper functioning. To the best of our knowledge, RADAR represents the biggest dataset for routing attacks in RPL. No previous dataset considered this amount of attacks in RPL.

RADAR contains five simulations for each attack considered. Implemented networks deploy 16 IoT devices and a single border router, belonging to a single DODAG structure. RADAR also contains ten legitimate simulations. For each simulation, NetSim allows to keep track of a packet trace file which contains a list of packets that have been exchanged throughout the simulation. Table 3.3 shows the average amount of packets that are recorded for simulations of different attacks. It is possible to notice that the amount of packets recorded on average depends on the attack considered. Attacks that aim at disrupting DODAG structure or influence control flow introduce high amount of packets. On the other hand, attacks that aim at diverting traffic or steal information, do not introduce heavy traffic, resulting in smaller simulation traces. For each of the packets recorded in the simulation, Netsim records the following features: packet type, application name, source, destination, transmitter and receiver identities, arrival and start time for application, network, data link, and physical layers, and payload size for the same layers. Source, destination, gateway, and next-hop IP addresses are also recorded along with the rank and version values for RPL control packets. All the features considered by NetSim can be extracted from un-encrypted network traffic. Un-encrypted mode is usually deployed in RPL-based networks thanks to its lightweight requirements and for the heavy constraints of IoT devices. On the other hand, even if encrypted versions of RPL are considered, our system would be capable of extracting the necessary information from the network traffic if the centralized server have the knowledge of security keys. This requirement is not heavy as the centralized server serves for the security of the network and its role can be compared to the role of certificate authorities. The knowledge of the packet trace file corresponds exactly to the deployment of sniffing devices capable of redirecting network traffic to the centralized IDS server.

RADAR contains simulations that last 1500 seconds each. Attacks are set to start at a random time between 500 and 700 seconds. The reason behind the attack start time is to be found in the setup time for most security systems. Indeed, RADAR is primarily meant and designed for IDS performance testing. To work properly, most security systems require some calibration time that should be attack free. We considered an interval of 500 seconds over a simulation of 1500 seconds to be long enough to satisfy most IDS’s calibration time requirements. Except for wormhole attack, one attacker was selected for each simulation. Attackers are selected specifically to show the effects of the attack on the network. Otherwise, an attack which does not affect the network in any way is not worth identifying for obvious reasons. For example, a blackhole attack on a leaf node would not drop any application packet. Therefore, its significance level would be null.

TABLE 3.3: RADaR’s simulated scenarios and the corresponding average amount of packets collected. For each packet specific features like source, destination, rank, etc. are collected.

| Scenario | Average number of packets | Average number of control packets | Average number of application packets |
|---------------------|---------------------------|-----------------------------------|---------------------------------------|
| Legitimate | 657K | 624K | 33K |
| Blackhole | 1.8M | 1.8M | 27K |
| Selective forward | 1.8M | 1.8M | 29K |
| Sinkhole | 1.6M | 1.6M | 33K |
| Continuous Sinkhole | 2.3M | 2.3M | 29K |
| HELLO flooding | 1.0M | 1.0M | 33K |
| Clone ID | 185K | 151K | 33K |
| Sybil | 185K | 151K | 33K |
| Wormhole | 257K | 218K | 38K |
| Version | 2.3M | 2.2M | 34K |
| Replay | 3.3M | 3.2M | 32K |
| Rank | 2.3M | 2.3M | 34K |
| Worst parent | 185K | 152K | 33K |
| DJS | 220K | 187K | 32K |
| Local repair | 2.3M | 2.2M | 35K |

To represent as precisely as possible real scenarios, IoT devices, in RADAR’s simulations, send application packets periodically with period of 1 second [36, 291]. Moreover, the pathloss in RADAR’s simulations follows a Friis free space pathloss model with exponent equal to 2. The given simulations settings allow RADAR to represent real scenarios faithfully while being reasonably sized. The extraction of RADAR was completed using a Windows 10 machine with 64 GB of RAM and an Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz processor. RADAR required around 400 hours of run-time to complete all simulations.

To summarize, RADAR’s characteristics are the following:

- RADAR contains packet trace files of 80 different simulations, with more than a million packets stored on average for each simulation.
- 14 well known attacks and legitimate scenarios are simulated. Five simulations for each attack (see Section 3.1.2) and 10 simulations for legitimate scenarios.
- Each simulation contains 16 IoT devices and a single border router that build a single DODAG structure. The considered devices are static, to recall most common RPL real world applications.
- Each IoT device forwards application packets with period of one second. This setup recalls RPL real world applications in which IoT devices periodically reports information to final users.
- Each simulation lasts for 1500 seconds. In attack simulations the malicious behaviour starts randomly between second 500 and 700.

3.3 Proposed RPL attacks detector: DETONAR

In this section, we propose DETONAR, a novel security mechanism to DETect rOutiNg Attacks in RPL. We first present an overview of the proposed mechanism in Section 3.3.1, followed by a detailed explanation of DETONAR’s pipeline: *traffic collection* in Section 3.3.2, *features extraction* in Section 3.3.3, *anomalies detection* in Section 3.3.4, *attack classification* in Section 3.3.5 and *attacker identification* in Section 3.3.6.

3.3.1 Overview

State-of-the-art detection mechanism introduces several challenges in RPL detector systems (see Section 3.1.3). We now summarize a list of properties that an RPL detector system should guarantee:

- P1 *No RPL-communication overhead.* RPL networks suffer from communications' overhead (see Section 3.1.2). The desired detection system should be an RPL-network independent entity, and it should not use RPL-communication channels.
- P2 *No RPL-nodes overhead.* The addition of operation in RPL-nodes increases their energy consumption. The desired detection system should not impact nodes' computational processes.
- P3 *Attacks Resistant* The desired detection system should face several RPL network attacks.
- P4 *Network Independent.* The desired detection system should work with different RPL network' topologies (i.e., nodes' connection, nodes' numerosity).
- P5 *Implementation flexibility.* Already existing RPL networks should integrate the detection system easily.

DETONAR's design aims to face properties P1-P5. However, in this work we do not focus on P4, while we test DETONAR only on small-size networks. We now briefly introduce DETONAR's pipeline, consisting of 5 steps, as shown in Figure 3.1.

1. *Traffic Sniffer* (Section 3.3.2). An ensemble of packet sniffers sense networks' traffic and forward it to a server. The sniffers are RPL-networks independent.
2. *Feature extraction* (Section 3.3.3). Extraction of a set of features describing the collected network traffic.
3. *Anomaly detection* (Section 3.3.4). A mechanism that analyzes nodes' traffic patterns to find potential anomalies.
4. *Attack classification* (Section 3.3.5). A signature-based mechanism that analyzes anomalies to identify potential attacks.
5. *Attacker Identification* (Section 3.3.6). A signature-based mechanism that identifies compromised nodes.

3.3.2 Traffic Collection

DETONAR employs sniffing devices to capture RPL networks' traffic. The optimal placement of these devices can follow [140, 24]. The collected traffic is then forwarded through secure channels (e.g., SSL) to an external server on-site or in the cloud. The transmission is conducted periodically, with time windows of a predefined size $\omega[s]$. ω is a DETONAR's hyperparameter. Information retrieved by sniffing devices contains general knowledge of packets exchanged by RPL devices. This information contains the type of packet sent (e.g., DIO, DIS, DAO, DAO-ACK, application), the address of the sender, receiver, source, and destination devices. If control packets are sniffed, additional information is considered as rank and version values.

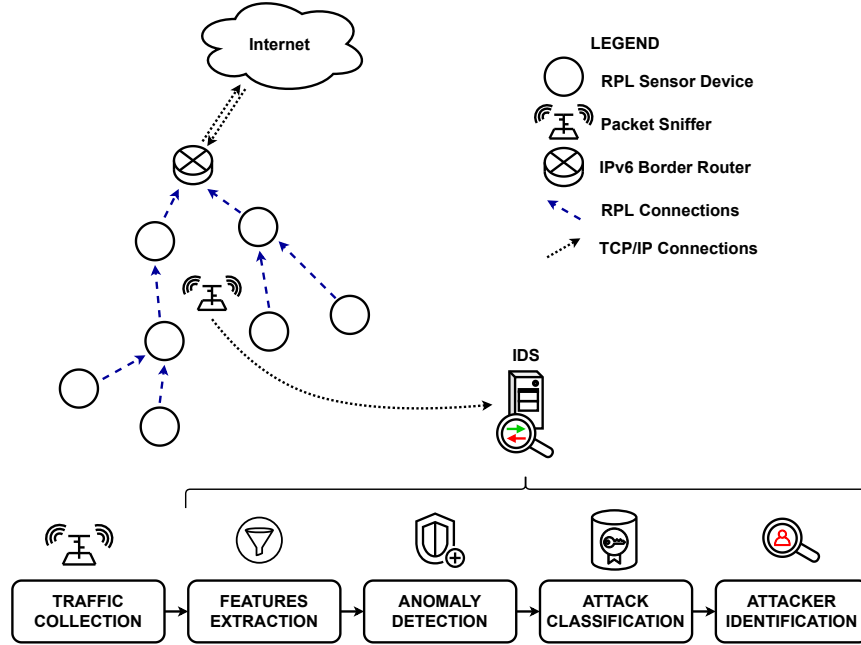


FIGURE 3.1: DETONAR deploys sniffing devices to sense the network traffic and forward it to the centralized IDS server. IDS server is in charge of detecting possible anomalies, attacks, and compromised devices.

Sniffing devices do not rely on the underlying RPL network to communicate the retrieved information with the server. Otherwise, the quality of the RPL communication would be affected by DETONAR’s workflow. Passive sniffing devices can communicate securely with the external server, leveraging various communication protocols (e.g., 4G, satellite). These protocols allow the safe deployment of DETONAR, introducing reasonable costs. The design and cost of specific communications between sniffing devices and external servers are not in the scope of this paper.

The introduction of RPL-agnostic sniffing devices allows us to achieve P1 and P2. Moreover, DETONAR achieves P5 since RPL-nodes do not require any additional computational effort (e.g., software update).

3.3.3 Features Extraction

DETONAR monitors each RPL node’s activities through the received traffic collected in the last time window. Formally, be \mathcal{N} an RPL network with $|\mathcal{N}|$ nodes and W_i^t the traffic collected at time window t for the i -th node n_i . Starting from W_i^t , DETONAR defines a set \mathcal{F} of 11 features as described in Table 3.4. \mathcal{F} describes quantitatively and qualitatively the sensed traffic. Features $f_1 - f_8$ are quantitative, as they express the amount of received or forwarded packets by each RPL device (e.g., number of forwarded/received DIO). Quantitative features give a measure of the traffic density that each node sustains. Features $f_9 - f_{11}$ are qualitative, as they express the considered node’s information (e.g., rank).

DETONAR represents W_i^t as a feature vector F_i^t

$$F_i^t = [f_1(W_i^t), \dots, f_{11}(W_i^t)]. \quad (3.1)$$

TABLE 3.4: Selected features of our IDS. It is possible to notice that each feature helps detect one or more attacks.

| # | Feature | Attacks Detected |
|----------|---|---|
| f_1 | Number of DIO received | HELLO flood, Local repair, Sinkhole, Continuous Sinkhole, Rank, Replay, DIS |
| f_2 | Number of DIO transmitted | HELLO flood, Local repair, Sinkhole, Continuous Sinkhole, Rank, Replay, DIS |
| f_3 | Number of DAO received | Worst parent, Sinkhole, Rank, Replay, Version |
| f_4 | Number of DAO transmitted | Worst parent, Sinkhole, Rank, Replay, Version |
| f_5 | Number of DIS transmitted | DIS |
| f_6 | Number of application packets received | Blackhole, Selective forward, Wormhole, Clone ID, Sybil |
| f_7 | Number of application packets transmitted | Blackhole, Selective forward, Wormhole, Clone ID, Sybil |
| f_8 | Transmitted vs Received applications rate | HELLO flood, DIS |
| f_9 | Rank | Sinkhole, Continuous Sinkhole, Rank, Replay, Local Repair |
| f_{10} | Version | Version |
| f_{11} | Next hop IP | Wormhole, Worst parent |

The extracted feature vector F_i^t is finally appended in the node n_i behavioral history B_i :

$$B_i = [F_i^0, \dots, F_i^t]. \quad (3.2)$$

Figure 3.2 shows the network representation schema. Each node n has its own history, with a different pattern across the various features.

Figure 3.3 shows an example of different patterns between features f_6 and f_2 (i.e., the number of received applications and the number of forwarded DIO) among three nodes (i.e., the root node, sensor 5, and sensor 12) at different depths of the RPL's structure in a RADAR simulation.

3.3.4 Anomalies Detection

As previously introduced, DETONAR relies on a hybrid approach: i) detection of anomalous traffic behaviors and ii) identification of the corresponding attack and compromised device(s). This section describes the anomaly detection stage, aiming to inspect each node's activities to find inconsistencies between its past and current status. This stage allows DETONAR to be resilient to different unknown attacks, as shown in [165]. With this component, DETONAR achieves P3.

Based on the extracted features in the previous step at time t (Section 3.3.3), DETONAR's anomaly detector inspects the minimum set of features $\mathcal{F}_A \subset \mathcal{F}$ that allows identifying the presence of the 14 attacks presented in RADAR. The set \mathcal{F}_A corresponds to: *number of DIO received*, *number of DAO transmitted*, and *number of applications received*. The usage of a minimum set of features allows DETONAR to reduce the number of false alarms and the computational cost.

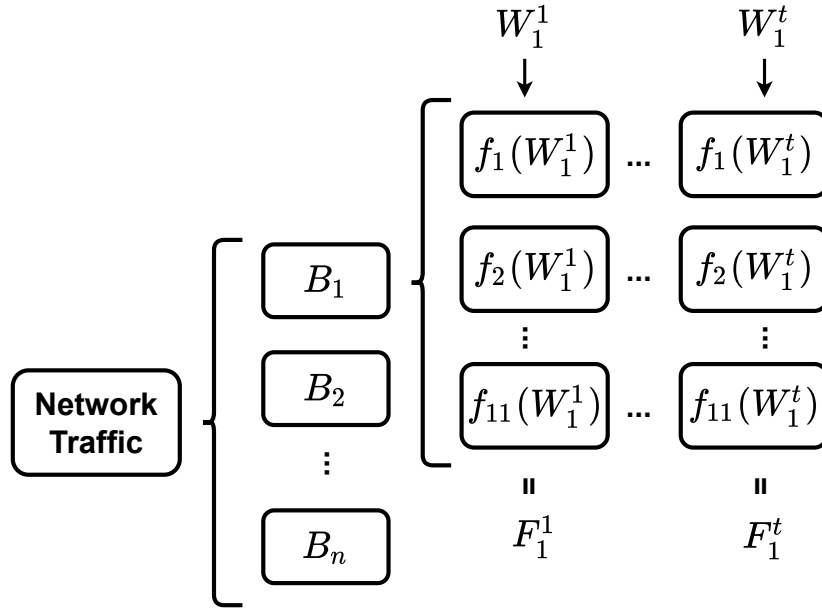


FIGURE 3.2: DETONAR feature extraction overview. Feature vectors F_i^t are extracted for each node i at each time window W_i^t as the composition of the 11 representative features selected. Node behaviour B_i is built as the composition of feature vectors F_i^t .

One of the challenges in RPL anomaly detection is the different traffic nature that each node has. To overcome this issue, DETONAR employs an anomaly detection algorithm $\mathbb{A}_{i,j}$ for each pair n_i, f_j in $\mathcal{N}, \mathcal{F}_A$. Formally, given the node n_i and its node behavioral history B_i , $\mathbb{A}_{i,j}$ analyses node's f_j history $h_{i,j}$:

$$h_{i,j} = [f_j(W_i^{t-\lambda}), \dots, f_j(W_i^{t-1})], \quad (3.3)$$

where λ indicates the history size that we consider. λ is a DETONAR's hyperparameters. In DETONAR implementation, $\mathbb{A}_{i,j}$ is an AutoRegressive Integrated Moving Average (ARIMA) model [35]. Being an autoregressive technique, ARIMA fits on previous values of the feature series to predict its future behavior. In ARIMA, an autoregressive mechanism is applied to the series rendered stationary via differentiation procedure. Autoregression (AR) and error moving average (MA) are leveraged to predict future values. Once the forecast is computed on the stationary process, the integration (I) operation is applied to compute the final prediction value. For an in-depth overview, we suggest [35]. ARIMA is a parametric function over: p , the number of autoregressive terms, d , the number of differentiation steps needed to make the series stationary, and q , the number of lagged forecast errors in the prediction equation. The selection of p , q , and d hyperparameters is fundamental for a correct fit. There exist automatic search algorithms that have been proposed to optimize such parameters. In our work, we applied a variation of the Hyndman-Khandakar algorithm [134], following ARIMA's implementation proposed in *Pmdarima*.³

ARIMA estimates $f_j^t(W_i^t)'$ using past node history $h_{i,j}$:

$$f_j^t(W_i^t)' \pm \mu = \mathbb{A}_{i,j}([f_j(W_i^{t-\lambda}), \dots, f_j(W_i^{t-1})], \alpha), \quad (3.4)$$

³<https://alkaline-ml.com/pmdarima/index.html>

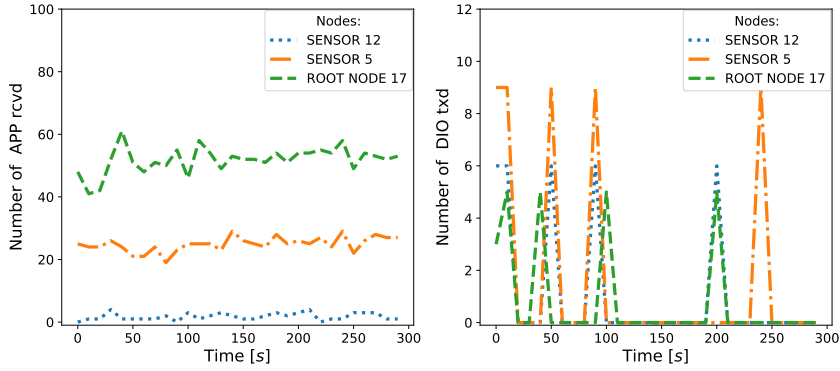


FIGURE 3.3: DETONAR extracts features from traffic received at a node to describe device's behavior. Feature series may differ significantly depending on considered feature and device.

where $f_j^t(W_i^t)'$ is the forecast value, and α is the confidence value (a DETONAR's hyperparameter); μ represent the prediction's boundary. DETONAR raises an anomaly for node n_i on feature f_j at time t if the following condition does not hold:

$$f_j^t(W_i^t)' - \mu \leq f_j^t(W_i^t) \leq f_j^t(W_i^t)' + \mu. \quad (3.5)$$

Figure 3.4 shows the anomaly detection mechanism applied to the same device and the same feature f_1 in a legitimate and an attack scenario. It is possible to notice that in legitimate traffic, the ARIMA raises no alarm. While, in the attack scenario, sinkhole produces an anomalous increment in the number of DIO packets received.

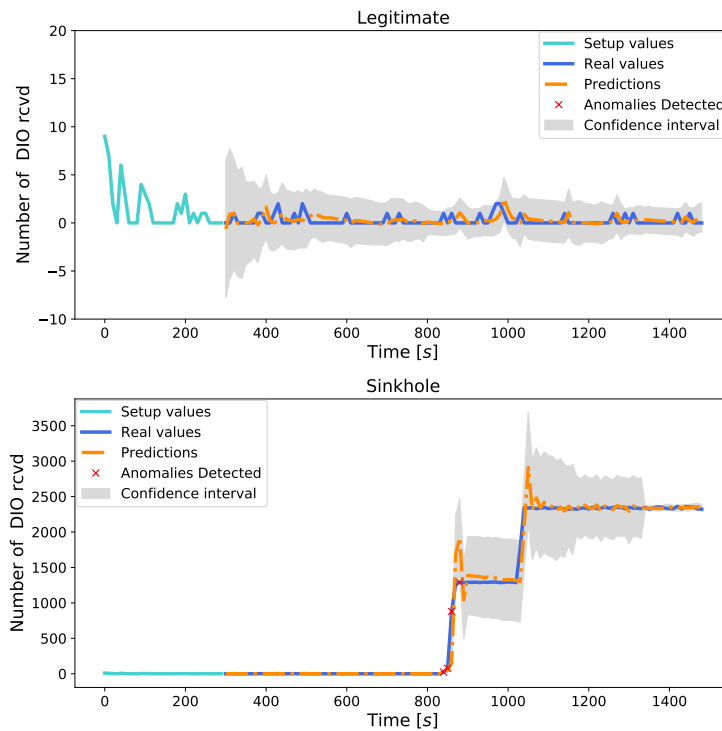


FIGURE 3.4: ARIMA applied to the number of received DIO packets of a device in a legitimate and a sinkhole attack scenario.

We remark that the use of ARIMA introduces the need for a setup period free of attacks. This setup phase's size depends on history size λ , a DETONAR's hyperparameter, and the time window size ω chosen for feature extraction. Another aspect to consider is the choice of one ARIMA per node and traffic behavior, which makes DETONAR limited in the scalability, and thus it does not achieve P4.

3.3.5 Attack Classification

The anomaly detection stage allows DETONAR to detect any potential attack leveraging the minimum amount of features. However, it presents a couple of drawbacks:

- *High false positives.* Any anomalous behavior causes ARIMA to raise an alarm. There exists no guarantee that an attack causes the alarm. Therefore, we need to filter out those anomalies caused by slight variations of legitimate behaviors.
- *No attack/attacker identification.* Alarms raised by the first stage of DETONAR simply signal that there exist some inconsistencies of behavior. These alarms do not allow compromised devices to be identified. Therefore, we need to introduce a classification technique that allows DETONAR to locate the attacker correctly.

To identify attack class, attacker identity and reduce the false positives, solving anomaly detection's drawbacks, we use several tests to profile the anomaly. In detail, once the anomaly detection mechanism raises the alarm, a list of suspected devices is produced. This list contains the devices which caused the alarm and their neighbors. Some attacks influence only features representing the neighborhood behavior while leaving attacker behavior features untouched. The list is passed to the attack classification mechanism, which is in-charge of classifying anomalies into different classes of attacks. This mechanism is implemented to make it possible to return false alarms whenever no attack is detected, allowing to identify and discard potential false positive alarms. To classify the attack correctly, DETONAR considers the 8 unused features of Section 3.3.3. These features consider the device's behavior and network structure. In particular, the DODAG is reconstructed at the centralized server due to the ability to record DAO packets exchanged between devices.

To classify anomalies into attacks and identify attackers, DETONAR implements a classification flowchart shown in Figure 3.5. For each node of the tree, a different rule (anomaly or signature-based) is applied to choose the path to follow. We now describe the different rules used by the classification mechanism:

- *Clone Identity - signature.* This rule compares active identities at W_i^t with the legitimate identities collected during the setup phase. In particular, we recall that DETONAR considers RPL devices sending application packets periodically (e.g. one packet per second), where each device can have different period. We define m as the maximum devices' period. Given the lossiness of RPL networks, DETONAR compares the identities between the setup phase and each possible sub-window of W_i^t of size $m \cdot c$, where the overlap is set to one second. Finally, DETONAR identifies a clone identity or a sybil attack if there is at least one mismatch among the comparisons. m represents an RPL network-related parameter, while c is a DETONAR's hyperparameter.
- *Changing DODAG - signature.* This rule checks if the attack impacts the DODAG. Since we are considering static scenarios, a modification of the

DODAG can only be due to an attack. Change in DODAG structure are detected comparing DODAG at time window W_i^t with its predecessor at time window W_i^{t-1} . This rule helps to identify two macro-categories of attacks: attacks on the DODAG, and attacks on the traffic.

- *Changing Rank - signature.* This rule allows to detect attacks that leverage the modification of the rank value. DETONAR filters the DIO packets sent by each device to check this rule, extracting the corresponding advised rank. A device advising a different rank in a static scenario is considered an attacker.
- *Changing Version - signature.* If no rank value has changed while the DODAG has been modified, advised versions are checked in this rule. Version values can be extracted from control packets in the same way as ranks. If a node has changed the version, then a version attack is detected. The corresponding attacker is identified as the first device which advised a new DODAG version. If no version and no rank values have been changed, but the DODAG was modified, then a worst parent attack is detected. The attacker is then detected as the device which changed the preferred parent.
- *Changing Transmitted Applications Rate - anomaly.* DETONAR checks if the considered node saw any change in the application packets that it transmitted. This rule is considered since attacks aiming at traffic can either manipulate application traffic or control traffic. Like the *changing DODAG* rule, this one helps to subdivide attacks against the traffic into attacks against application traffic and attacks against control traffic. This rule consists of an anomaly-based detection scheme. Indeed, DETONAR applies ARIMA on the series of transmitted application packets. This approach is identical to anomaly detection, but it changes only the considered feature.
- *Children Changing Destination - signature.* If an attack against application traffic is detected, the proposed rule checks if any node is changing its next-hop. If this happens, then the node changing next-hop is considered to be part of a wormhole attack. With respect to the *state-of-the-art* the proposed approach is the simplest enabling the detection of wormhole attacks. No considerations regarding devices' positions or power of transmission is done, and the resulting performances are surprising.
- *Incoming vs Outgoing Traffic - anomaly.* When no change in next-hop is detected, our IDS checks the ratio between received and transmitted application packets. In legitimate scenarios, this ratio's trend should remain almost constant. Instead, in blackhole and selective forward attacks, it decreases significantly. DETONAR applies anomaly-based detection scheme (i.e., ARIMA) to identify possible anomalies in this ratio sequence. If an anomaly is found, then the attacker is the device analyzed by the ARIMA. Otherwise, a false alarm is sent.
- *Produce New Control Packets - signature.* When no attack against application traffic is detected, the proposed rule checks if any node produces unnecessary control packets. The forged control packets may be either DIO or DIS. The same check is done for DIO and DIS control packets. DETONAR considers the number of control packets sent by a suspected device in the last time window W_i^t . If this value is bigger than the previous maximum value of control packets transmitted in a time window W_i^j with $j \in [1, t - 1]$, then an attack is detected. Otherwise, a false alarm message is raised, as there exists no significant proof

of an attack on control traffic. Mathematically, an HELLO flooding attack is detected, if $DIO(W_i^t) > \max\{DIO(W_i^{t-1}), DIO(W_i^{t-2}), \dots\}$. Where the DIO function counts the number of DIO packets transmitted by a device i in the time window W_i^t .

The combination of the proposed rules allows DETONAR to detect the most known attacks against RPL. Moreover, the proposed mechanism is flexible against new attacks. New rules may be added to classify novel attacks upon their discovery.

3.3.6 Attacker Identification

A security system should automatically identify attacker's identity to remove the malicious device from the network. DETONAR introduces specific rules to identify the attacker location from the attack class identified following Section 3.3.5. The knowledge of attack class is sufficient to identify the attacker correctly since the attacker behavior of different attacks is specific. We now present the attacker identification mechanism that is used for each class of attacks:

- *Clone ID and Sybil attacks.* As already mentioned, DETONAR's check on active devices immediately identifies the attacker's identity. The device missing from communication corresponds to the attacker's original identity.
- *Attacks on rank.* This class of attacks comprehends sinkhole, local repair, rank, continuous sinkhole, and replay. To find the attacker's identity DETONAR checks which is the first device that advised a changed rank. One effect of these attacks is to change multiple nodes' ranks. Therefore, to find the attacker, it is necessary to find the oldest change in rank values. This approach is possible since the proposed IDS knows each device's rank from the sniffed DIO packets.
- *Version attack.* In this case, the approach to identify the attacker is very similar to the attacks on rank. The proposed IDS identifies the attacker as the first node that advised a different version in a DIO packet.
- *Worst parent attack.* To identify the attacker in the worst parent attack, DETONAR checks what device changed next-hop IP in the time window W_i^t . This straightforward principle effectively detects the attacker in a complex scenario like the worst parent attack. DETONAR can use such a simple principle due to the attack classification mechanism that relies on more complex decisions.
- *Wormhole attack.* DETONAR's attack classification mechanism detects those devices that changed next-hop IP and transmitted an anomalous amount of application packets in wormhole scenarios. This approach by itself allows the proposed IDS to find attackers' identities. Attacker devices are the only nodes that satisfy the two conditions presented above.
- *Blackhole and Selective forward attacks.* For these two attacks, DETONAR detects an attack only for those devices that drop an anomalous amount of application packets. Therefore, DETONAR's attack classification mechanism is already identifying attacker identities.
- *HELLO flooding and DIS attacks.* In HELLO flooding and DIS scenarios, the malicious device sends an anomalous amount of control packets. DETONAR's attack classification mechanism already identifies the attackers as those devices transmitting an anomalous amount of control packets.

Figure 3.5 shows the rules for attackers' identification.

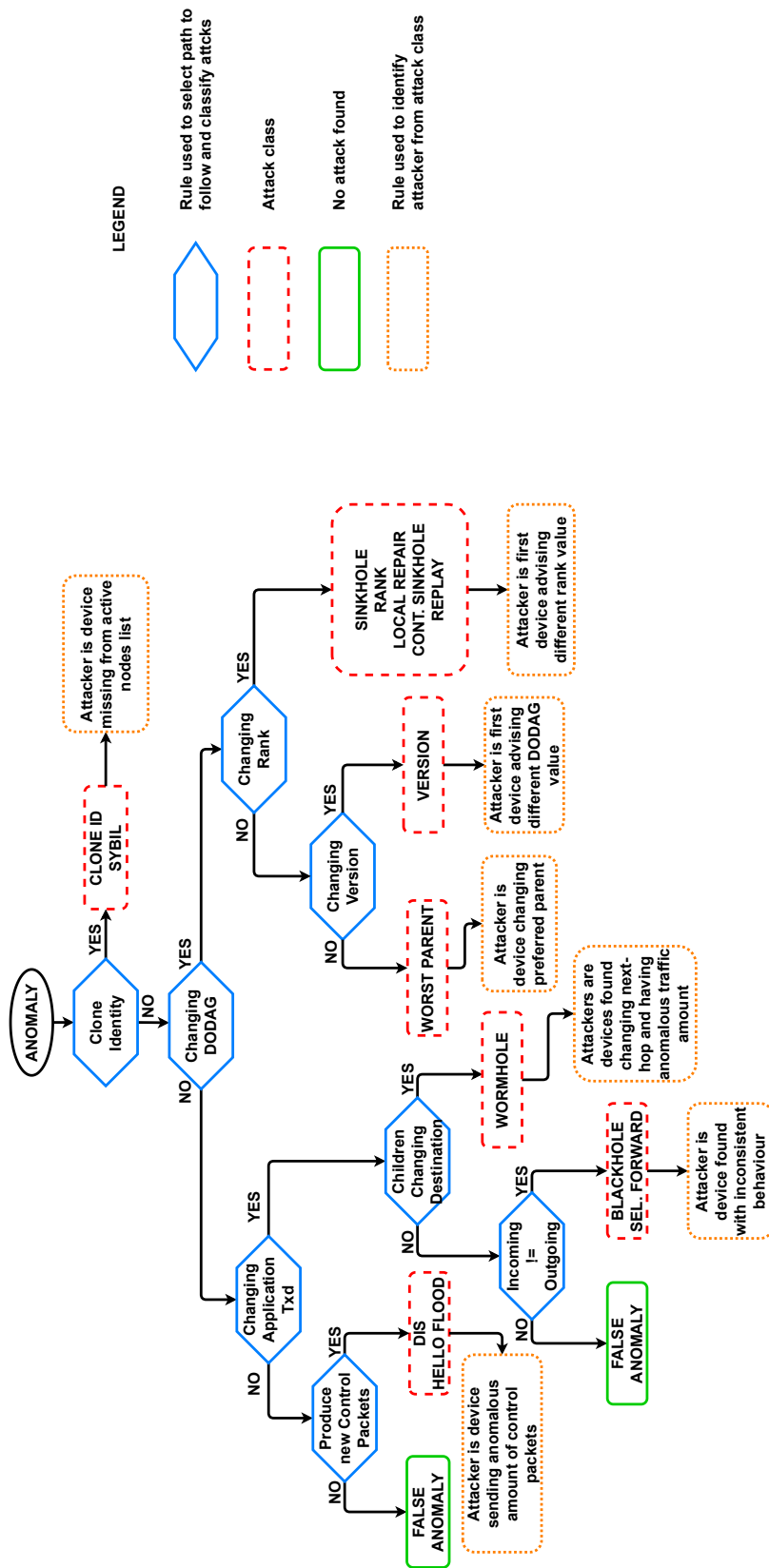
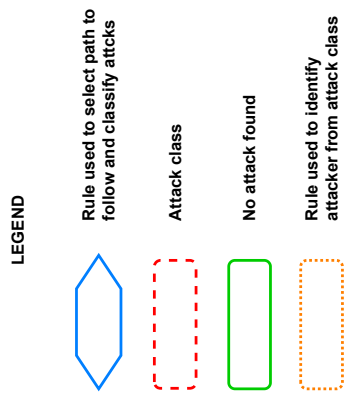


FIGURE 3.5: DETONAR’s decision flowchart. Each rule is based on considerations regarding RPL networks that help classify the final attack. Attacker identity depends on the attack classified.



3.4 Implementation and Performance evaluation

This section first presents the implementation details of the DETONAR mechanism (see Section 3.4.1). Finally, in Section 3.4.2 we present the results obtained using DETONAR on RADAR dataset. In particular, we present DETONAR’s true detection percentage, false positives, and the computation time needed by the anomalies detection step.

3.4.1 Implementation & Hyperparameters tuning

We make our implementation of DETONAR publicly available. DETONAR’s scripts take as input a packet trace corresponding to a simulation and return as output DETONAR’s runtime information. This information contains the anomaly detection mechanism’s decision at each time window W_i^t . If the anomaly detection algorithm raises the alarm, then the attack classification results are also present in the runtime information. Running DETONAR’s scripts from the packet trace files belonging to RADAR corresponds to the real-world application of packet sniffers that communicate with the centralized server.

We now describe the DETONAR’s hyperparameter tuning among the time window size ω , the test significance α , and the history size λ (see Section 3.3). Tuning is conducted on a training set containing five legitimate simulations. The goal of the tuning is to minimize the false positives (FP) of DETONAR. Figure 3.6 shows the false positives for different ω , α , and λ values. To select ω , we fix $\lambda = 30$ and $\alpha = 10^{-4}$. ω affects with different trends different features, not giving statistical relevant results. We set $\omega = 10$ due to computational performance reasons only. To select λ , we fix $\alpha = 10^{-4}$ and $\omega = 10$. We can notice that when increasing ARIMA history size, we reduce the false positives. To reduce the setup time as much as possible while allowing DETONAR to detect eventual attacks accurately, we set $\lambda = 30$. We fix $\lambda = 30$ and $\omega = 10$ to select α ’s value. A small α lead to small FP. We thus set $\alpha = 10^{-4}$. Finally, concerning c , we recall that in RADAR devices are deployed sending one application packet every second, i.e., $m = 1$. Therefore, we set $c = 3$.

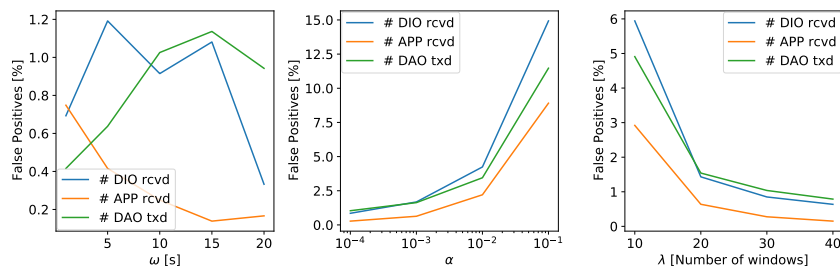


FIGURE 3.6: ARIMA’s false positives are influenced by DETONAR’s time window size (i.e., ω), history size (i.e., λ) and test significance (i.e., α).

3.4.2 Results

We now present DETONAR’s performance computed on our proposed dataset, RADAR. We remark that we do not compare DETONAR with state-of-the-art IDS since they are not implemented using NetSim and their re-implementation is not

³<https://github.com/AndAgio/DETONAR>

trivial. We evaluate three aspects: false positives, attack and attacker(s) detection accuracy, and finally, time performance. We need to make some considerations for the attack detection accuracy. We know the attack’s starting time τ_{Att} and the compromised device(s) for each simulation. We consider an attack to be detected correctly if DETONAR raises the alarm and classifies the attack correctly after τ_{Att} . Instead, the attack is considered misclassified if DETONAR does not raise any alarm or raises an anomaly correctly, but it does not classify the correct class of attack. We are bound to consider these metrics for classification’s performance for the following reasons:

- Some attacks start at time $\tau_{Att} + \epsilon$ since they need the reception of a specific packet to be triggered. For example, in sinkhole attack, the attacker waits for the reception of DIO packets to trigger the publication of forged rank value. No assumption can be made on the duration of ϵ . Depending on the attack considered, the size of the network and the simulation time ϵ may vary significantly. Therefore, it is impossible to identify the actual attack starting time.
- No assumption can be made on the label of packets in the network traffic, since the attacks can indirectly affect the performance of non-victim devices. For example, sinkhole attack induces the attacker’s neighbors to change their rank, and the attacker’s neighbors advise new rank values in forged DIO packets. The aftermath is in a complex labeling process, which we avoid.

False Positives Performances

To analyze FP, we test DETONAR over five legitimate simulations (separate from the five used for tuning) and measure the number of identified attacks (~ 2000 predictions). In detail, we measure the FP in both anomaly detection and attacker identification stages (see Section 3.3). Table 3.5 shows the FP rate results. In particular, the anomaly detection stage based entirely on ARIMA has high FP, while the second stage with both anomaly and rule-based signature pushes the score close to zero.

TABLE 3.5: False positives for Anomaly Detection (AD) and AD + Attack Classification (AC) of DETONAR in five legitimate simulations.

| Simulation ID | AD | AD + AC |
|---------------|------------------|------------------|
| 6 | 2.13% | 0% |
| 7 | 1.83% | 0.10% |
| 8 | 3.06% | 0% |
| 9 | 2.08% | 0.05% |
| 10 | 5.19% | 0.05% |
| Overall | 2.86% \pm 1.24 | 0.04% \pm 0.04 |

Detection and Identification Performances

We test the ability of DETONAR to identify the attack and attacker(s) over five simulations for each of the 14 attacks presented in RADAR. Table 3.6 summarizes the detection performance. DETONAR successfully detects with 100% of accuracy 8 out of 14 attacks; in these attacks, the attacker is always successfully identified. DETONAR seems to suffer only blackhole, continuous sinkhole, and local repair attacks. Concerning blackhole attack, we notice that in some simulations, the attack affected nodes with few application packets, resulting in challenging detection. In continuous sinkhole and local repair, instead, DETONAR misses the detection only for those

TABLE 3.6: DETONAR can detect most attacks with satisfactory accuracy both in terms of attack detection (DET) and attacker identification (ID). In wormhole, ID contains the number of attackers identified correctly out of the two existing.

| Attack | 1 | | 2 | | 3 | | 4 | | 5 | | Overall | |
|---------------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|---------|------|
| | DET | ID | DET | ID | DET | ID | DET | ID | DET | ID | DET | ID |
| Blackhole | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | | | 60% | 60% |
| Selective Forward | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | 100% | 100% |
| Sinkhole | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | 100% | 100% |
| Continuous Sinkhole | ✓ | ✓ | | | ✓ | ✓ | ✓ | ✓ | | | 60% | 60% |
| HELLO Flooding | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | 100% | 100% |
| Clone ID | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | 100% | 100% |
| Sybil | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | 100% | 100% |
| Wormhole | ✓ | 2/2 | | 0/2 | ✓ | 2/2 | ✓ | 2/2 | ✓ | 1/2 | 80% | 70% |
| Version | ✓ | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ | ✓ | 80% | 80% |
| Rank | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | 100% | 100% |
| Replay | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | 100% | 100% |
| Worst Parent | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | ✓ | | 80% | 60% |
| DIS | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | 100% | 100% |
| Local Repair | | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | 40% | 100% |

simulations in which the attack does not produce any change in the DODAG structure; in these scenarios, DETONAR identifies reasonably a hello flooding attack.

Time Performances

Finally, we analyze DETONAR’s time performance. We find the computational bottleneck in ARIMA models (see Section 3.3, anomaly detection stage), which are applied for each window W_i^t . We measure ARIMA performance on 170 thousand predictions using a standard laptop (i.e., Intel Core i5-3230M CPU, 8 GBs DDR3 RAM). The average prediction time is 1.1 seconds. Given this performance, and the possibility to distribute the computation among several cores, we can state that DETONAR can be deployed in real-world small-scale networks.

3.5 Summary and Future work

In this work, we implement and detect 14 well-known routing attacks against RPL in IoT networks. Using the various network logs obtained while simulating these attacks, we build our RADAR dataset. RADAR represents the largest and most significant dataset of routing attacks against RPL. We believe that the availability of such a comprehensive dataset is a step forward in the research field of IoT security. Based on the RADAR dataset, we propose a novel and complete security mechanism called DETONAR, capable of detecting 14 well-known attacks. The simulation results show that DETONAR provides excellent attacker identification results (i.e., low false positives) with no RPL communication overhead, thanks to its sniffing approach. DETONAR does not require any heavy computation or firmware modification at IoT devices, which makes it a practical solution. It also introduces future flexibility as, upon discovering novel attacks, one can modify the attack classification mechanism by adding new rules to previously unknown attack rules. Finally, DETONAR’s flexibility allows its quick deployment on the underlying network, as it does not require any IoT devices update. In the future, we plan to do a more

in-depth analysis of DETONAR concerning the following aspects: (i) test its performance on a real-world testbed, (ii) investigate its performance in dynamic networks, (iii) extend its attack detection algorithm to generalize features behaviours among different devices, (iv) test its performance on large-scale networks, and (v) compare its performance with state-of-the-art IDS implementation using NetSim.

Part II

The role of cybersecurity in web web platforms

Chapter 4

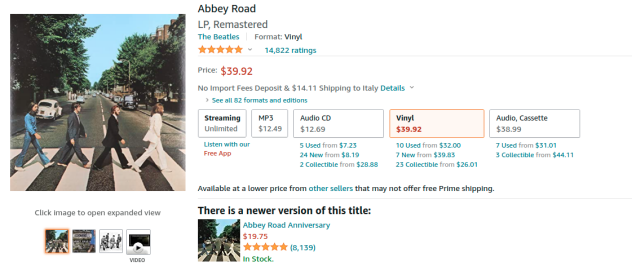
A Novel Review Helpfulness Measure based on the User-Review-Item Paradigm

Online Review Platforms (RPs) provide valuable information about others' opinions about products they bought or experiences they had. Thus, if someone considers having dinner at a restaurant, s/he can first look at other people's opinions and then decide accordingly. Today, there are specialized RPs for diverse entities: hotels and restaurants (e.g., Yelp, Trip Advisor), products (e.g., Amazon, E-bay), and movies (e.g., IMDb). An example of a product on Amazon product and its top review is shown in Figure 4.1. The number of reviews is increasing: TripAdvisor had 200M reviews in 2014 and a billion in 2021 [273]. As a single item may have thousands of reviews, there is a critical need to find the reviews that users might find most helpful. Moreover, reviews directly impact product sales [58, 83], making the need to identify helpful reviews critical for the integrity of reviewing platforms.

Much work has been done to estimate a review's helpfulness score. In particular, prior works primarily used *user judgment* as a ground truth: given a review, users can express feedback about whether the review was helpful or not. Such scores can be expressed as numerical values, e.g., "5 out of 10", meaning that among ten votes, five were positives. Leveraging such a "helpfulness ratio" (HR), we can generally find three types of predictors [78]: *regressors*, whose aim is to predict the exact HR, *classifiers*, whose aim is to predict if a review is helpful (i.e., the HR exceeds a threshold) or not, and *rankers*, whose aim is to order reviews from the most to the least helpful. In general, most past works extract features from the review text or the context in which the reviews are written.

Literature limitations Our work started with the following question: what makes a review helpful? We investigated the literature in this area, and noticed that answering this question is difficult. Indeed, it nontrivial to even clearly understand the state of the art. The problem was first raised by Diaz et al. [78], where the authors identified three major causes:

1. *A historical problem* as studies goes back to early 2000. Furthermore, the analyses were conducted on different datasets, most of which are not publicly available. This limits the reproducibility of past methods, adversely affecting efforts to compare current work with past ones.
2. *Lack of proper comparisons*, as newly proposed models are not always compared with the state of the art baselines.



(A) Product.



(B) Review.

FIGURE 4.1: Example of an Amazon’s product and its top review.

3. *Lack of evaluations*, where the effect of new proposed features is not adequately investigated (e.g., with ablation studies).

Though hundreds of papers have been written on this topic, the above three flaws make it a herculean task to compare past efforts carefully. Diaz et al. [78] make an important first attempt in this direction. A second important survey by Du et al. [82] investigates the predictive effect of a large set of state-of-the-art features. The study is conducted on the open-source dataset Amazon 5-cores [122].

Contributions Our goal is to carry out the most comprehensive survey of past work on review helpfulness surveys. While the three limitations listed by Diaz et al. [78] (and described above) are valid, the *lack of publicly available code* is another significant drawback of past work. This is a major issue, especially when custom features are extracted from a dataset. We found that, in general, prior studies textually (and usually briefly) described the extraction phase, leading to many possible interpretations. For example, suppose we wish to compare an approach that only textually describes the feature extraction and works on a dataset that is not publicly available. In that case, the replication of such a baseline might be prone to errors. A second (but no less critical) major issue relates to the *model validation strategy*: indeed, most of the works validate their model with k -fold cross-validation. This can be erroneous for temporal data (reviews are timestamped) because a training fold might contain reviews from a future time t_2 compared to some reviews in a test-fold which may be posted at a time $t_1 < t_2$). In this case, such efforts might use information from the future (t_2) to train on and predict the past (t_1). Moreover, the helpfulness score

of a review may be related to past reviews written on the same product. Zhou et al. demonstrated that the order in which reviews are written could influence whether a review is marked as helpful or not [331]. Intuitively, a product might have thousands of reviews, and thus a well-written review might not be noticed. We address such validation issues by adopting a proper temporal splitting strategy.

We further propose a novel way to describe the helpfulness of a review based on the User-Review-Items (URI) paradigm. Most past works approach the helpfulness prediction task with a review-centric model, i.e., the features adopted in the baselines only describe the review item. URI, on the other hand, focuses on the platform as a whole. We demonstrate how useful information can be gleaned from the context in which the review is published (e.g., relationship with past product reviews). In a study using six Amazon categories datasets, we validate URI using 270 models and show that classifiers gain +5 points in F1 scores on average using URI.

Our contributions can be summarized as follows:

- We propose the *User-Review-Item* (URI) paradigm, a graph-based representation of Review Platforms. We use this representation to organize the space of possible features derivable from RPs.
- We carefully summarize 20 years of prior work on the topic.
- We present an open-source feature extractor framework that implements most of the features discussed in the helpful reviews literature and derivable from Amazon reviews. We release code (i.e., feature extractor, training procedures) and processed data upon publication.
- We empirically demonstrate the advantages of the URI framework with an extensive analysis considering six Amazon datasets [122], with a total of 270 trained models involved. On average, models gained +5 percentage points in their F1-score. We further investigate several challenges of the task.

Organization This chapter is organized as follows. Section 4.1 presents our vision of features that describes a User-Review-Item (URI) paradigm. Section 4.2 summarizes 20 years of prior work on the topic. Section 4.3 describes our experimental settings, as well as our feature extraction framework. Section 4.4 shows how temporal dynamics can impact different features. Section 4.5 shows the performance of different classification algorithms. Section 4.6 studies whether deep neural networks outperform simple baseline classifiers. Finally, sections 4.7 and 4.8 describe related works and conclusions, respectively.

4.1 The Users-Reviews-Items (URI) Paradigm

Section 4.1.1 first introduces our User-Review-Item (URI) paradigm and shows how this can help organize the vast number of features studied in past work. We then describe the features according to their URI category: Section 4.1.2 presents user graph features, Section 4.1.3 presents review graph features, Section 4.1.4 presents item graph features, and Section 4.1.5 presents multi-graph features. It concludes Section 4.1.6 with a visual summary figure of these features.

4.1.1 The User-Review-Item (URI) Paradigm

In this paper, we represent an RP via a graph consisting of three subgraphs: one each about users, reviews, and items being reviewed as shown in Figure 4.2. We call

this the *User-Review-Item* (URI) paradigm. Nodes in a URI might have two types of connections: *intra-connections*, if the connection is between two nodes of the same type (e.g., two users), and *inter-connections*, if the connection is between two nodes of different types (e.g., from a user to a review).

Intra-connections can be defined in many ways. For instance, two users might be connected if their Jaccard similarity w.r.t. products they both reviewed exceeds a bound. Two reviews might be connected if sentence-level embedding vectors have a similarity exceeding a given bound [45]. Two products might be connected if they are similar (e.g., two cell phones might be connected).

We define *Inter-connections* in two ways: *users-reviews connections* and *reviews-items connections*. The nature of *users-reviews connections* might vary based on the review platform. The linkage may be defined by the relationship users have with reviews, e.g., “authored” (the user posted a review), or “voted” (the user judged a review). Similarly, we might have different type of links among reviews and items, e.g., *item-based* (the reviews belongs to the same item), *category-based* (the reviews belong to similar items). We do not discuss user-item connections in this paper since such links can be derived by combining information about *users-reviews connections* and *reviews-items connections*.

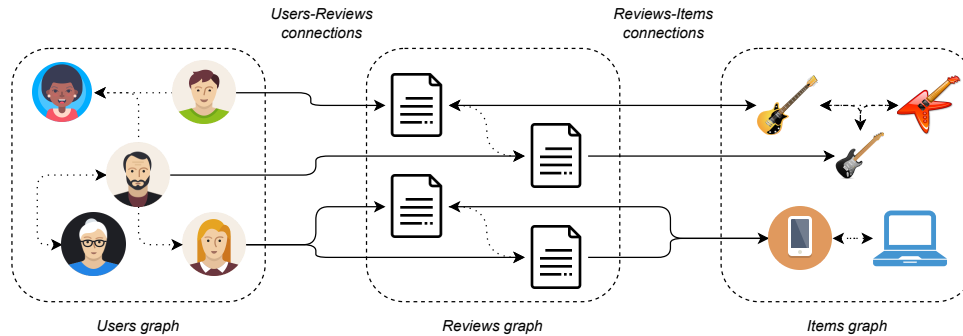


FIGURE 4.2: This figure illustrates the User-Review-Items (URI) Paradigm of a review platform.

Formally, a URI contains the following nodes:

- *User Graph*. Nodes in the user graph consist of registered users of the RP. Users have one of two roles: reviewing an item and voting whether a review is helpful or not. Furthermore, user nodes can have connections. The nature of the connection varies based on the RP or on the knowledge we have of user interactions. For example, in Amazon, a connection can be drawn if a user u_i votes for a review written by the user u_j ; however, only Amazon has this level of knowledge. On the other end, in RPs like Ciao, users can create explicit relationships with one another (e.g., a user u_i follows a reviewer u_j).
- *Review graph*. Nodes are reviews in the RP. Generally, a review has several meta-features, such as the review’s title, star rating, and textual description. Different RPs may offer different metadata: a common example is when multiple features of a product are evaluated via star ratings. For example, a restaurant may get star ratings for food, service, ambiance, and more. A review graph might link reviews together via edges, e.g. if a review mentions or replies to another review.
- *Item graph*. This is the set of items that can be reviewed in an RP. Different RPs may offer different types of items. For example, on Amazon, we might

evaluate products we can buy. On Yelp and TripAdvisor, we might review restaurants and hotels. On IMDb, we might evaluate movies. The items graph may connect two items using different rules, e.g., two items are connected if they belong to the same brand, or the same category (e.g., genre for movies).

From each individual graph (i.e., user, review, and item graphs) and type of connection (i.e., intra or inter), we can derive features that may depend heavily on the RP. For example, the Amazon and IMDb platforms may share some common features derivable from their reviews (e.g., review length, star rating), but may also have unique features (e.g., Amazon review product rank, film location on IMDb). In the rest of the section, we summarize features adopted in prior baselines using a URI organization. The organization follows the following hierarchy: *graph-level - item level* (i.e., nodes or edges) - *feature category*.

Before discussing the various features in detail, we first comment on the popularity of various feature families. In particular, we analyze the distribution of features described in prior works we describe in Section 4.2. The Sankey diagram in Figure 4.3 shows that $\sim 70\%$ of features come from the review graph. This implies that most past works focused on features derivable from review text without considering the RP information carefully.

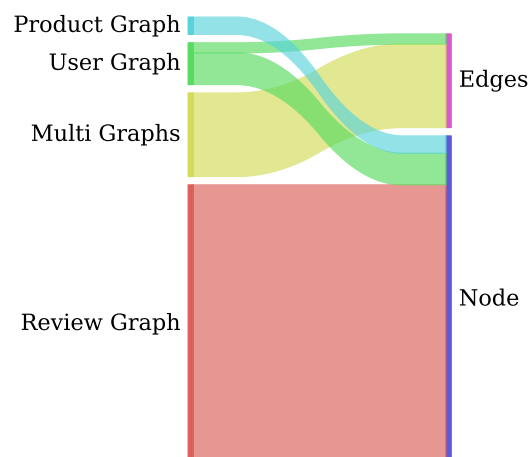


FIGURE 4.3: Feature families popularity in prior works.

4.1.2 User Graph

We now describe the features derivable from the URI user-level subgraph.

Node-level

Identity Disclosure Identity disclosure-based features include boolean variables for the following: if the reviewer uses a real name, has a real profile photo, has a nickname, hobbies, birthday, location, a link with his/her webpage, interests, biography, and the logical or of the above information [102, 239, 331].

Reviewer Rank The reviewer rank provided by an RP indicates the reliability and quality of authored reviews [102, 187, 284, 168, 331]. This information is often displayed in reviews or in reviewer profiles. Some studies found ranking to be linked

to review quality [284], but others disagree [168]. Different types of features can be derived from the reviewer rank, e.g., if a reviewer is a top k reviewer. Using this knowledge, we can derive binary variables indicating if reviewers are listed in top10, top50, top100, and top1000 [102]. Similarly, Yelp shows elite reviewers [331].

Edge-level

Social Connections In RPs, users interact with each other. For example, u_i might like (or dislike) someone else's review. Similarly, an RP might allow social connections, such as a friendship between two users. While potentially useful, this category of features is highly dependent on the RP. For instance, Amazon can keep track of a helpfulness vote (i.e., u_i likes or dislikes a review written by u_j); however, an external viewer cannot see this.

Several past studies used Ciao as the RP as it explicitly shows user-user links [284, 187]. Using Ciao, Lu et al. derived users' in-degree and out-degrees [187]. Tang et al. [284] captured the relationship between voters (i.e., users judging reviews) and review authors. They hypothesize that users tend to vote the same way as their connections. Voters and reviewers may also share similar preferences. A preference context metric is introduced to capture such similarities. A few Yelp studies used the number of friends as a proxy for a reviewer's reputation [239, 331].

4.1.3 Review Graph

In the case of review graphs, we only discuss node-based features — we saw no use of edge based features in such graphs. We categorize node-based features into three families.

- *Content*. Set of properties about the review's text (e.g., sentiment, emotions).
- *Style*. Set of properties that capture stylistic characteristics of the written review (e.g., description length).
- *Meta-features*. Set of properties derivable from the metadata (e.g., star rating).

Content

Content-based features can be expressed via patterns, embeddings, and emotions.

Content-based patterns. Such features capture the presence of specific words or topics in a sentence. A popular approach uses the Bag-of-Words and its variants, like binary occurrence [213], term occurrence [213], term frequency [213], and TF-IDF statistics based on unigrams [156, 82, 213, 317] or bigrams [156, 82]. At a paragraph level, Liu et al. counted the number of paragraph separators (e.g., pros, cons, the good, the bad) [181].

One would expect a good review to describe specific characteristics of the reviewed products. For example, when reviewing a *digital camera*, a desirable topic of interest might be the quality of the *zoom*. Defining interesting properties of a product a priori is not trivial, and different works counted the number of product properties cited in a review text as a feature [156, 125, 181]. Similarly, we can count the number of sentences containing product-related information [181]. Here, the authors estimate both the number of brands in reviews, number of brands in titles, number of products in the review, and number of products in titles. Finally, a review

topic can be modeled using a Latent Dirichlet Allocation [33] representation which is a topic distribution [82]. Liu et al. [184] uses a *document profiling* model to identify product features contained in a review; this approach is built on top of the one presented by Lim et al. [178]. The authors then extract several features: the number of products referenced in the review, number of product features per sentence, number of sentences with at least one product feature, number of product features divided by the number of sentences with at least one product feature, and number of sentences with at least one product feature divided by the number of sentences in the review. Similarly, Min et al. count the number of product mentions [205].

Content-based features are related to the RP or the class of products being reviewed (e.g., hotels, restaurants). For example, Mahony et al. quantified the number of review-template questions (e.g., “would you recommend this hotel?”) answered in the review [218]. Furthermore, the authors also quantified the number of optional personal details and purpose of visit details contained in the review. Dominant terms contained in a given review are a feature proposed are investigated by Tsur et al. [294]. They propose an unsupervised review ranking based on the similarity between the dominant terms in a given review and an “ideal” perfect review vector. The intuition is that a review should contain some terms that describe the product, and the closer to an ideal review, the better.

Word embeddings. Word embeddings are a powerful representation based on deep neural networks that converts a textual review into an embedding vector. Du et al. [82] use two types of embeddings: skip-gram with negative sampling [202] and global vectors [229]. The review embedding is defined as the average of the review word embeddings.

Emotion-based. These features capture feelings and emotions in reviews. As stated by Du et al. [82], state of the art systems primarily use one of the following tools: Linguistic Inquiry and Word Count dictionary (LIWC) [228], General Inquirer (GI) [277], Geneva Affected Label Coder (GALC) [257], Opinion Lexicon [130], Senti-WordNet [19], SentiStrength [288], and VADER [133]. Such tools capture sentiments and/or emotions. Many tools merely count the number of words in a category of words associated with an emotion. For example, if a selected tool determines if a sentence is positive or negative, the result is a bi-dimensional vector. The first dimension is the number (or ratio) of positive words contained in the review. The second dimension does likewise for negative words. Emotional vectors are adopted in many works [196, 317, 82, 86, 187, 181].

Another direction is to measure emotions by extracting the number of objective and subjective sentences in a given review [184, 102, 329].

Style

Stylistic features describe how a review is written. We identified four major groups: *char statistics*, *length*, *readability*, and *syntax statistics*.

Char statistics This family of features identifies specific patterns of characters that might be associated with the utility of a review. Examples of such features are the number of exclamations marks [156, 317], number of HTML bold tags [156], number of HTML line breaks [156], the ratio of special tags (i.e.,
, <p>) [219], the

ratio of upper and lower characters to other characters in the review text [218], the ratio of upper to lower characters in the review text [218], percentage of upper and lower characters [219], percentage of upper characters [219], ratio of capitalized sentences [187].

Length-based features These features capture the depth of a review. The hypothesis is that useful reviews should contain a lot of information — if so, users would prefer longer reviews to shorter ones. The length can be captured in several ways. Popular metrics are the number of words [156, 218, 196, 82, 184, 168, 102, 161, 239, 331, 219, 86, 131, 317, 187, 181], number of characters [82, 168, 223], number of sentences [156, 196, 82, 184, 102, 219, 317, 187, 181], average sentence length [156, 82, 184, 168, 102, 219, 317, 187, 181], average number of characters per word [168], number of paragraphs [181], average paragraph length [181].

Readability. We expect that a good review will be well-written and easy to read. It is thus necessary to capture *readability* aspects. Readability measures can be captured with metrics such as the Flesch–Kincaid Reading Ease score, Flesch–Kincaid Grade Level, Gunning Fog Index, Simple Measure of Gobbledygook, Automated Readability Index, and Coleman–Liau Index. These features are adopted in several baselines [196, 82, 161, 219].

Readability can be further measured by counting the number of one-letter words, the number of two to nine-letter words, and the number of words with ten or more letters [168, 219]. Analogous to the readability score features, the review’s grammatical correctness might play a role in whether a review is considered helpful or not. Grammatical correctness can be measured by counting the number of misspelled words [82, 184, 102] or the number of misspelled words divided by the total number of characters in the review [168].

Syntax statistics. These features capture properties of the review syntax such as the percentage (or the count) of question sentences [156, 317, 82] and the percentage (or the count) of exclamatory sentences [82]. Lu et al. [187] defined the vocabulary richness of a review as the ratio of unique words to the number of words in a review.

Part-of-Speech (PoS) tokens are often used. For example, we can quantify (e.g., percentage, count) the number of nouns, verbs, and adjectives contained in a review [156], or quantify (e.g., count, percentage) specific parts of speech [156, 196, 184, 82, 205, 329, 125, 187, 183]. Hong et al. [125] hypothesized that volitive auxiliary words might be linked to unreliable reviews. However, the authors did not find any correlation between this feature and review helpfulness. On the other hand, they found tense-based features to be linked to review helpfulness.

Meta-features

We can also extract features from review metadata. We identify two groups of features: *miscellaneous* (i.e., generic features) and *star rating* related features.

Miscellaneous. This family contains meta-features that are strictly related to the reviewing platform used. For example, in the case of TripAdvisor, we can extract two features that indicate whether the reviewer responded to optional content in the review, e.g., if the reviewer completed optional liked and disliked forms of

the review [218]. Zhou et al. [331] defined a boolean variable if the review contains pictures. Temporal features like the number of elapsed days since the review was posted are used in many studies [223, 239, 168]. Different works used Calendar ranges to create a one-hot variable for the year of the review (e.g., if the dataset contains reviews written between 2005 and 2008, then we have a 4-values vector) [105, 331].

Star rating. During the review process, RPs usually ask users to evaluate further the product with a star rating (i.e., a numerical score). For example, on Amazon, the reviewer can use a scale from one to five stars. Star-rating (or valence) based features are the star rating itself [156, 196, 161, 211, 218, 223, 239, 236, 86, 105, 131], or properties that describe the star rating, such as if the rating is moderate (i.e., three stars) [102].

4.1.4 Item Graph

We can also define features based on the item-item graph.

Node-level

Meta-Features RPs may show products with several properties. We now report on some of the *product info* features. A famous example is the *product type* [211, 168, 223], i.e., a variable that tells if the product is an experience or a search good. While found significant by Mudambi et al. [211], the authors also claim that categorizing products into these two categories is not trivial. Price information can be used in two forms [168, 102]: the manufacturer’s suggested price and Amazon’s retail price. Product popularity is another feature that can be retrieved on many RPs. For example, on Amazon, we can extract the sales rank of the product [168, 102]. Zhou et al. defined a dummy variable for the restaurants available in the dataset [331]. Similarly, Chen et al. collected product information (e.g., price, rank, shipping days) on different dates, e.g., every day since the product release day [54].

4.1.5 Multi Graphs

We now discuss features based on multi graphs. This set of features only consider connections (i.e., edges) among different types of nodes.

User-Review Connections

The knowledge of users’ past reviews might help in predicting the helpfulness of their future or current reviews. Mahony et al. [218] defined a set of features retrievable from users’ past reviews, i.e., mean and standard deviation of helpfulness scores of past reviews and the percentage of reviews that received at least T votes. Similarly, we can extract the number of reviews written by the user and the average and standard deviation of the number of reviews written by all users in the RP. Such features are used in many baselines [218, 168, 102, 213, 239, 331, 131, 187, 236]. Ngo-Ye et al. [213] proposed a recency feature, which is defined as the difference between the current review date and the date of the last review posted by the reviewer.

Lu et al. [187] proposed the average star rating (excluding the current review) of reviews rated by the reviewer. This is motivated by the fact that reviewers might have different reviewing approaches (e.g., someone might be more likely only to

give positive scores). Similar features were used by Mahony et al. who extracted the average and standard deviation of sub-scores of hotel reviews [218]. Liu et al. [183], in a study focused on IMDb movie reviews, captured reviewers' expertise with radial basis functions. In particular, the authors exploited past reviews of movie genres to build such knowledge.

Review-Item Connections

The overall star rating of a product, defined as the average of the star ratings of past reviews, is often used in prior works. For example, Chen et al. used such an average as a feature, and several variants derivable from it, such as the average star rating of those reviews with more than 80% helpful votes, the average number of reviews written by top1000 reviewers of the platform, and the average star rating of spotlight reviews (i.e., displayed in the product's page) [54].

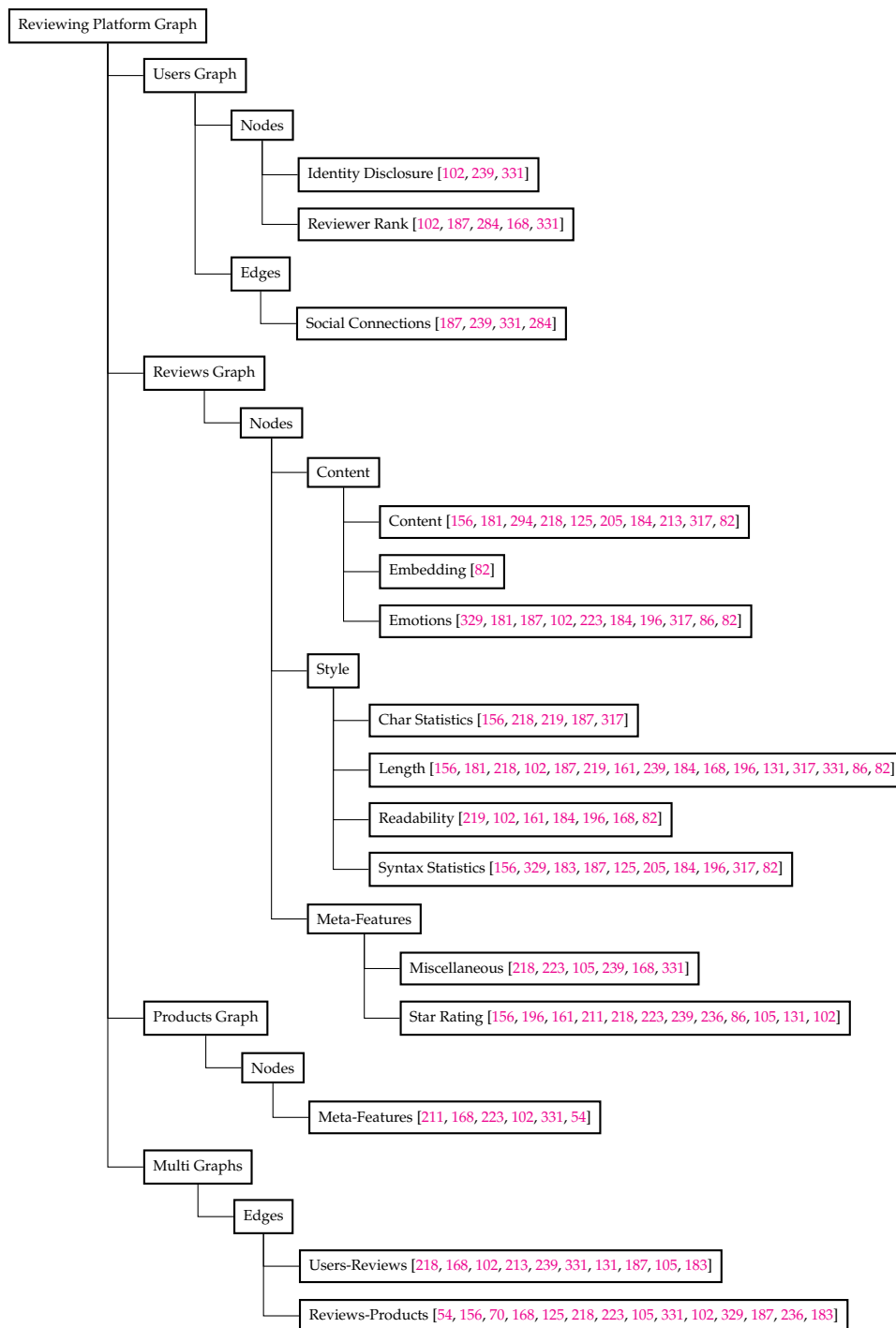
When considering the product's average star rating and a current review of the product, we can extract divergences. Some works measured the divergence as the absolute value of the difference between the current review rating and the average product rating [156, 70, 168]. Danescu-Niculescu-Mizil et al. used a signed version of divergence [70]. Similarly Hong et al. studied the divergence between the current review's sentiment and the sentiment in past reviews, where the sentiment is extracted with WordNet [125].

From the product's history, we can extract the number of reviews for a given product and the average and standard deviation of the number of reviews written for all the products on the review platform [218, 223, 54]. Some works used temporal information like the elapsed time since the product release [54, 168]. The number of days since the first review was posted [105, 331] has also been used as a feature. Several works also extracted information about the product's reviews from the product history. An example is the number of past reviews [168, 102] and the average review rating over time [168, 102].

Another important aspect highlighted is the review order [105, 331], i.e., how the reviews are organized and shown to readers. In this study, the review order negatively impacts the helpfulness score. For example, it seems natural to think that the first displayed review can be observed (and thus voted on) by more users than those not displayed on the first page of reviews.

By considering the history of all products, a plagiarism-based tool can identify nearly-identical review pairs [70]. Zhang et al. [329] argued that a review r should differ from the official product specification s and the editorial review e . Thus, they compute the similarity between a given review and the specification/editorial information. This measure is based on the cosine similarity among the TF-IDF of the two variables, i.e., $sim(r, s)$ and $sim(r, e)$. Lu et al. [187] proposed a conformity measure defined as the KL divergence between a given review and unigram language models of past reviews. Liu et al. [183] used a radial basis function to model the decay time of a given product review with the idea that, after a certain time, it is unlikely that a review will be considered helpful.

4.1.6 Summary



4.2 Prior Works

This section provides a chronological summary of relevant prior art.

Kim2006 [156] Kim et al. studied Amazon reviews for MP3 players and digital cameras, with the overall goal of ranking reviews based on usefulness. They developed an SVM regression-based ranking system that, given a product and its reviews, predicted the helpfulness score. This study primarily used structural features. They

found that review length (i.e., number of tokens), n-gram TF-IDF (i.e., unigram), and the star rating are important predictive factors.

Zhang2006 [329] This work studied Amazon reviews and predicted helpfulness using support vector regression and linear regression. They used Part-of-Speech statistics and counts of the emotion level in the review (e.g., as a count of subjective-objective words). The results show that the best model uses shallow PoS statistics, implying that the utility of a review is highly dependent on its linguistic style.

Liu2007 [181] This study first filters low-quality reviews using an SVM trained on three sets of features: *informativeness*, *readability*, and *subjectiveness*. The experiments are conducted on Amazon reviews. The authors further discuss three biases. (1) *Imbalance vote*: when considering ground truth derivable by the RP (e.g., Amazon), they noted that users tend to vote positively rather than negatively. (2) *Winner circle bias*: only a few reviews get a large number of votes (positive and negative). (3) *Early bird bias*: older reviews tend to have higher votes.

Chen2008 [54] This study analyzes the impact of reviews on Amazon's new books. Prior to this paper, most research focused on the aggregate numerical review scores while leaving important questions (e.g., the relationship between helpful reviews and reviewers) unexplored. Three measures are considered: the quality of the content (i.e., the helpfulness score), the reviewer's reputation, and the impact of the reviews displayed on the product page (spotlight reviews). The findings can be summarized as follows: (1) higher ratings are associated with higher book sales, (2) products with a large number of helpful reviews have a stronger impact on consumer purchase decisions, (3) spotlight reviews have a stronger effect on sales, and (4) no evidence was found that links reviewer reputation with sales.

Liu2008 [183] This study of IMDb reviews presents a predictor that combines three factors: the reviewer's expertise, the writing style of the review, and the timeliness of the review. The authors highlight three major issues in predicting the helpfulness: (1) Recent product reviews tend to have no votes (or few votes), making it difficult to predict helpfulness of such reviews (early bird bias [181]). (2) There is a "monopoly effect" if only highly ranked reviews are shown to users by the RP, preventing even helpful new reviews from being shown (winner circle bias [181]). (3) The review labeling process (i.e., if a review is helpful or not) can be affected by wrong voting (e.g., spam voting [142]).

Danescu- Niculescu-Mizil2009 [70] This seminal 2009 study of Amazon reviews suggests that the helpfulness of a review depends both on the review and on other reviews of the same product. The authors investigated four hypotheses.

1. *Conformity hypothesis*. A review is considered helpful when its star rating is close to the majority.
2. *Individual-bias hypothesis*. A review is helpful if the opinion is in agreement with the reader's opinions.
3. *Brilliant-but-cruel hypothesis*. Negative reviewers are considered more reliable than positive ones.

4. *Quality-only straw-man hypothesis.* Review helpfulness depends only on review-related features.

The first three hypotheses were analyzed by observing the discrepancy between the review's star rating (i.e., the one to predict its helpfulness) and the product's average star rating (i.e., the product being reviewed by the review we want to predict helpfulness).

The *conformity hypothesis* does not hold since the signed value of the difference between the review's star rating, and the product's average star rating is not symmetric. For example, the authors considered two groups of reviews with signed deviation equal to -2 and +2 and found that the median values of their helpfulness ratios are different. Similarly, the authors highlight that reviews with positive star ratings tend to have higher helpful scores, which is in contrast to the *brilliant-but-cruel hypothesis*: i.e., users positively evaluate more, on average, positive reviews.

To distinguish the *individual-bias* hypothesis from the *conformity hypothesis*, the authors analyzed the signed star rating deviation, where the products are grouped by the variance of star rating. The idea is to find cases where the opinions (e.g., star rating) do not come from the same single-peak distribution. The authors observed that as the variance increases, the distributions move from a single hump (low variance) to two humps (high variance). The authors validated the *individual-bias hypothesis*.

We conclude with the *quality-only straw-man hypothesis*. The concept is straightforward: if a review is considered helpful for a product, a second nearly-identical review for a similar product should have a nearly-identical helpfulness score. Nearly-identical pairs of reviews were discovered using *plagiarism* tools. The authors further group such pairs based on the discrepancy between their star ratings. A statistically significant difference between the helpfulness ratios of members of such pairs indicates the influence of non-textual factors on the helpfulness score. The authors showed that the *quality-only straw-man* hypothesis does not hold and that there are factors unrelated to text that affect the review helpfulness score. This finding is particularly relevant since it implies that studies that consider only text-related features might not be adequate.

O'Mahony2009 [218] Similarly to [156], Mahony and Smyth designed a system in 2009 to recommend the most helpful reviews for a given TripAdvisor hotel. The proposed JRiP classifier is trained over features derivable from the review text (e.g., length) and from the RP (e.g., number of reviews written by the author, number of reviews written for a hotel). The authors find reputation-based features (e.g., average user helpfulness score) to be the most significant ones.

Tsur2009 [294] This study presents *RevRank*, an unsupervised algorithm for selecting the most helpful book reviews. The algorithm retrieves a set of reviews, identifies a lexicon of dominant terms, and orders the reviews according to a similarity metric. Since the algorithm is unsupervised, the ground truth (i.e., the ratio of helpful votes to total votes) is unnecessary. Given a set of reviews and an externally balanced corpus, RevRank defines a VC (Virtual Core) review. The VC can be seen as the best review that we can derive from the given set of reviews. The VC is computed using a set of dominant terms contained in the reviews. Finally, RevRank computes the similarity between each review vector and the VC vector, and it ranks the reviews accordingly.

Lu2010 [187] This work extends the set of features used by prior works (mainly textual features) with contextual information about reviewers. The authors use a semi-supervised learning technique that uses a small amount of labeled data and a large amount of unlabeled data. The approach can be applied even when information about reviewers is missing information.

Mudambi2010 [211] This study develops a predictive framework around three factors: the product type (i.e., search goods or experience goods), the review length, and the star rating. The research, conducted on 1587 Amazon reviews, presents two outcomes. (1) The product type denotes different helpfulness patterns, e.g., for services, moderate reviews (i.e., reviews with a star rating that is not extreme) are more helpful, while this is not true for search goods. (2) Review length has a positive correlation with the helpfulness score. This effect is stronger for search goods.

O'Mahony2010 [219] This paper considers structural and readability features and evaluates them on Amazon and TripAdvisor reviews. The authors find that the impact of these two types of features is stronger on Amazon than on TripAdvisor.

Ghose2011 [102] This paper's contribution is twofold: (1) different factors that impact review helpfulness, and (2) the economic impact (e.g., product sales) of reviews. The authors study review-based features (e.g., length, readability), information that can be retrieved from the RP, such as the reviewer history (e.g., number of past reviews), reviewer identity (e.g., name, bio, hobbies), and product history (e.g., number of past reviews). The authors adopted a Random Forest Classifier to predict both impacts on sales and helpfulness score. The researchers find that: (1) The more subjective reviews are, the higher the impact on sales. (2) For some products, the higher the readability of reviews, the higher the impact on sales. (3) Reviews including both objective and subjective opinions tend to be more helpful. (4) Reviews with low readability (and a high number of grammatical errors) tend to have lower helpfulness scores. (5) Historical reviewer information (e.g., average past helpfulness score) have a mixed effect on helpfulness score.

Pan2011 [223] This paper analyses the impact of different factors on helpfulness scores and finds that review length and star rating have a positive effect on the helpfulness score, but product type can play a confounding role. The results are similar to those presented by Ghose et al. [102].

Godes2012 [105] This paper investigates the sequential and temporal dynamics associated with online reviews and considers Amazon book reviews as a case study. The authors propose (1) a sequential dynamic, defined as the sequence of reviews ordered by posting time, and (2) a temporally dynamic, defined as the amount of time since a book is released (and available for review). The results suggest that both dynamics are impactful factors in online reviews.

Hong2012 [125] This paper develops new features linked to *user preferences*. Three categories of user preference features are developed based on the following hypotheses: (1) users prefer reviews that meet their information needs, (2) users prefer credible reviews, and (3) users prefer reviews that follow the mainstream opinion. The authors use Amazon reviews to show that these new features improve prediction performance.

Korfiatis2012 [161] This work studied the interplay between review helpfulness and review content. Their analysis is built on top of three elements: *conformity*, i.e., a review is considered more helpful if its star rating is close to the majority score, *understandability*, i.e., a review is considered more helpful when it is easier to read, and *expressiveness*. The authors' experiments on Amazon reviews suggest that the readability of reviews has a greater effect on the helpfulness ratio of a review rather than other characteristics such as the review length.

Min2012 [205] The authors proposed a novel metric to rank product reviews by "*mentions about experiences*". This metric tries to capture user experiences with the reviewed product. The idea is that reviewers who provide details about their experiences are more reliable than others that do not.

Preadeep2012 [239] This study shows that feature sets that include both reviewer and review characteristics are highly correlated with the helpfulness of Yelp reviews.

Qiu2012 [236] This paper studies the role of aggregated ratings (e.g., the average number of stars) in RPs. It shows that the credibility of reviews decreases as the difference between a given review's rating and the average rating increases. The authors find a stronger effect when the review has a positive score.

Liu2013 [184] This study looks at review helpfulness from the product designer's perspective. Its findings suggest that customers' and designers' perspectives differ; thus, building classifiers based on the standard helpfulness ratio based on customer votes is not appropriate for designers.

tang2013context [284] This study examines the hypothesis that reviews may not be equally useful to different users. The authors collected a set of graph-based contextual features from RPs, and they modeled the problem as a recommendation system. The experiments conducted on Ciao reviews suggest that context-based features are a valuable source for the helpfulness prediction task.

Lee2014 [168] This paper explored the power of neural networks using engineered features to represent Amazon reviews. They used a standard Multi-Layer Perceptron (MLP) trained over 20 features, including product data, review characteristics, and the textual characteristics.

Martin2014 [196] This work is based on the hypothesis that emotions in reviews may be linked to helpfulness. The authors compared the predictive power of emotion-based features with baseline features proposed in past works (i.e., length, Flesch Reading Ease, and PoS counters). Using data from Amazon, Yelp, and TripAdvisor, the authors show that emotion-based features outperform all baseline features except for part-of-speech-based features.

Ngo-Ye2014 [213] This paper used Amazon and Yelp reviews to show that using features that capture reviewer engagement (e.g., reputation, commitment, and current activity) along with review-based features improves predictive performance.

Filieri2015 [91] This study uses a questionnaire where participants describe their behavior when reading reviews. Each question requires answers on a 7-point Likert scale ranging from strongly disagree to strongly agree. The findings suggest that users are primarily influenced by the review quality, followed by customers overall ratings.

Huang2015 [131] This study considers both quantitative measures (e.g., review length) and qualitative aspects of reviewers (e.g., reviewer experience and impact). The study uses two Amazon datasets. The first dataset contains random reviews of specific products, while the second has reviews written by the top 10000 reviewers. The authors found that: (1) the length of a review is important but only to a certain extent — but this variable is not important w.r.t. top Amazon reviewers; (2) when considering top reviewers, factors like reviewer experience or impact are not significant for helpfulness; (3) star rating is a significant factor.

Yang2015 [317] This study tries to define a helpfulness classifier that can apply to different datasets and RPs. The authors argue that by mixing text-based features with non-textual ones, there is a risk of making the model less transferable. Using experiments on Amazon data, the authors argue that transferability is enhanced by considering just two specific sets of interpretable semantic features.

Zhou2017 [331] This paper highlights the factor that most prior studies assume that reviews are mutually independent. Using Yelp data, the authors show that, in general, the order of reviews is negatively related to the helpfulness score.

Chen2018 [49] This paper studies the limitation that prior work required massive amounts of ground truth for each domain of interest. They also considered the fact that “out-of-vocabulary” words might be encountered, i.e., words related to products’ category. The paper uses Convolutional Neural Networks (CNNs) based on words and characters and show, using Amazon data, that their approach is more transferrable than previous efforts.

Diaz2018 [78] This survey paper provides an excellent overview of the area. The authors discussed the difficulty of “organizing” prior research and determining the state of the art. The authors identified two possible causes: (1) different datasets (most not publicly available) were used in prior work, and (2) many efforts are not built on top of prior work. They make several recommendations:

- *Task*: if the goal is to determine if a given user will find a review to be helpful, the helpfulness score might not be appropriate, since users that voted for reviews might not be aligned with a specific user’s expectation;
- *Data*: current publicly available datasets might not be optimal for the helpfulness prediction task. Instead, datasets that contain more information about voters and reviewers might be needed.
- *Baselines*: when new algorithms are proposed, all prior features and models should be considered and evaluated.

Eslami2018 [86] This study investigated features linked to review length, sentiment polarity, and star rating to maximize the review helpfulness. Using an Amazon and Insureye datasets, the authors reported that helpful reviews have medium length, low review score, and negative or neutral sentiment.

Du2019 [82] This survey[82] studied a wide set of features derivable from “review nodes”. Using an SVM classifier trained on six Amazon 5-core datasets [122], the authors experimented with different combinations of features. The best classifier used all the extracted features.

Fan2019 [89] This study argues that many past efforts ignore essential metadata, such as the title, the brand, the category, and the description. The authors proposed an end-to-end deep neural network to model the task. Using Amazon and Yelp data, the authors show that their approach outperforms baselines.

Qu2020 [237] This paper uses (1) review sentiment and (2) the fact that a reader might pay more attention to particular parts of a review to develop a neural network-based predictor that includes an attention mechanism.

Qu2020 [238] The authors propose CA-GNN (Category-Aware Graph Neural Networks), a GNN-based model for Taobao. The authors leverage GNNs to capture the relationships between users and reviews of an RP.

4.3 Experimental Settings

We first present the features we extract (Section 4.3.1), and the datasets (Section 4.3.2) used in our experiments.

4.3.1 Feature Extraction Framework

In this section, we present the framework we used to extract URI-based features from Amazon reviews [122]. Table 4.1 shows the families of features used — we now go into the details of these features.

TABLE 4.1: Description of features available with our framework.

| Group | Dim | Users Graph | | Reviews Graph | | Item Graphs | Multi Graphs |
|--------------------|-----|-------------|-------|---------------|-------|-------------|--------------|
| | | Nodes | Edges | Nodes | Edges | Nodes | Edges |
| Product History | 3 | | | | | | ✓ |
| Product Metadata | 2 | | | | | ✓ | |
| Review Embedding | 700 | | | ✓ | | | |
| Review Emotions | 140 | | | ✓ | | | |
| Review Metadata | 3 | | | ✓ | | | |
| Review Readability | 12 | | | ✓ | | | |
| Review Structure | 41 | | | ✓ | | | |
| Reviewer History | 4 | | | | | | ✓ |

Product History Features When a new review is posted, we need to understand the review in the context of earlier reviews of the same product. Given a product p , the history H_p is a sequence of reviews ordered by their posting time:

$$H_p = \{r_0, r_1, \dots, r_n\}, \quad (4.1)$$

where r_n is the last posted review. For each review $r_i \in H_p$, we extract a set of features describing older reviews, i.e., $\{r_0, r_1, \dots, r_{i-1}\}$. Given a review r_i , we extracted three features: the number of previously posted reviews, their average star rating, and the divergence between the current review's star rating and the aforementioned average star rating.

Product Metadata We used 'product price' and 'product rank' which are available in the Amazon Reviews dataset [122].

Review Embedding The text of each review is converted to embedding representations, following [82]. We extracted a topic representation using LDA, fixing the number of topics to 100. The implementation follows [148] and uses Gensim [246]. We also computed two 300-dimensional vectors using GLOVE (i.e., glove-wiki-gigaword-300 version) and Word2Vec (i.e., word2vec-google-news-300) representations. We set the embedding vector for a review to be the average of word embeddings associated with words in the review.

Review Emotions Following [82], we extract a set of emotional features. We start with the GALC lexicon [257], which contains a dictionary of words associated with 38 emotions. For example, 'annoyed' is linked to the 'irritation' emotion. Similarly, words like 'fine', 'good', or 'nice' are linked to a 'positive' emotion. We decompose sentences into a list of stemmed tokens, and then count how many tokens belong to each emotion. Some tokens might not belong to any particular emotion. We defined another counter that expresses the 'unknown' emotion for these cases. Each emotion count is then divided by the total amount of tokens processed. Thus, a review is expressed as a GALC feature vector containing 39 values (38 emotions plus "unknown").

Sentiment-based features are derived using Opinion Lexicon [130] (OL), SentiWordNet [19], and VaderSentiment [133]. We used NLTK [186] to extract a positive and negative sentiment score. The extraction follows a procedure similar to the one described with GALC since the NLTK Open Lexicon contains two sets of negative and positive words. From NLTK, we further extract the SentiWordNet [19] features: positive, negative, and objective valences. Again, the extraction procedure follows the one described above. Finally, we extract negative, positive, and neutral valences of Vader sentiment¹. We also used 93 LIWC2015 93 features.²

Review Metadata We use the star rating (1 to 5 scale) and define a feature that says if the star rating is moderate [102] (3 or not). We also used the elapsed time since the review is posted as a feature.

¹<https://pypi.org/project/vaderSentiment/>

²As LIWC is a proprietary software, we do not make it part of our open-source extraction tool.

Review Readability We counted the number of misspelled words within a review using Hunspell³. We then computed the ratio of the number of misspelled words and the total number of characters in the review. We also counted the number of misspelled words in a review using an extended version of Hunspell containing Wikipedia titles (i.e., `enwiki-latest-all-titles-in-ns0`). This feature ensures that brands, technologies, etc. which are not part of standard English, are also considered. We also extracted features that describe the complexity of reading a review. We count the number of words containing only one character, from two to nine characters, and more than ten characters. Using Textstat⁴, we measure several readability metrics like Flesch–Kincaid Reading Ease score, Flesch–Kincaid Grade Level, Gunning Fog Index, Simple Measure of Gobbledygook, Automated Readability Index, and the Coleman–Liau Index.

Review Structure We measure the length of a review as the number of NLTK tokens, sentences, number of characters, average sentence length, and average characters per word. We further count the number of sentences that are questions and sentences that are exclamations. Finally, we compute the count and percentages of different Part-of-Speech tags.

Reviewer History We define features to capture a reviewer u 's past behavior. We can define u 's activities as a sequence of past written reviews ordered by their posting time:

$$H_u = \{r_0, r_1, \dots, r_n\}, \quad (4.2)$$

where r_n is the last written review. Thus, for each review $r_i \in H_u$, we extract the following describing the reviews written earlier by the author, i.e., $\{r_0, r_1, \dots, r_{i-1}\}$: the number of reviews, the average star rating, the sum of past helpful votes received, and the average of votes (positive and negative) received.

4.3.2 Dataset

For our experiments, we used the Amazon review dataset [122]. In particular, we use both reviews 5-cores and product metadata of the following six categories: Cell Phones and Accessories, Digital Music, Electronics, Pet Supplies, Toys and Games, and Video Games. The ground truth is defined as:

$$\text{helpfulness ratio} = \frac{\#\text{positive votes}}{\#\text{total votes}}. \quad (4.3)$$

We consider a review *helpful* if the helpful ratio exceeds 0.75. We only considered reviews with 3 or more votes in our training, validation, and testing sets. We split each category into training (70%), validation (10%), and testing (20%) sets using a temporal approach (i.e., based on reviews dates).

Table 4.2 summarizes statistics about the different datasets. For each category and split, we report the number of samples and the percentage of helpful reviews. First, we see that in all cases (excluding the Digital Music category), the percentage of helpful reviews decreases from the training to the validation and testing sets. Second, the ground truth is generally imbalanced toward the helpful class (excluding

³<https://pypi.org/project/cyhunspell/>

⁴<https://pypi.org/project/textstat/>

the ‘Video Games’ category). Third, the percentage of helpful reviews varies by category. For example, ‘Pet Supplies’ category has 75% of training samples belonging to the helpful class, while the same number is 48% in the ‘Video Games’ category.

TABLE 4.2: Dataset statistics. We show dimension and percentage of helpful reviews for each set.

| Category | Train | | Val | | Test | |
|-----------------------------|-------|----------|------|----------|-------|----------|
| | Dim | Help [%] | Dim | Help [%] | Dim | Help [%] |
| Cell Phones and Accessories | 11918 | 65.49 | 1689 | 54.35 | 3421 | 56.56 |
| Digital Music | 20009 | 53.98 | 2868 | 52.82 | 5725 | 47.46 |
| Electronics | 54018 | 70.05 | 7714 | 54.36 | 15446 | 51.28 |
| Pet Supplies | 11735 | 75.88 | 1669 | 67.75 | 3361 | 68.34 |
| Toys and Games | 16070 | 72.31 | 2321 | 59.89 | 4598 | 56.31 |
| Video Games | 55062 | 48.59 | 7891 | 34.28 | 15739 | 29.65 |

4.4 Temporal Analysis

Each of the datasets (cf. Section 4.3.2) used in our experiments was temporally split into training, validation, and testing sets using a temporal approach. The oldest reviews belong to the training set, followed by more recent ones in the validation set. The newest ones represent the test set. This splitting strategy mirrors real-world usage: we train a model on past reviews to predict if a future review is helpful or not. This splitting strategy is also essential in light of [331]’s result that the order of reviews matters.

Section 4.3.1 presented eight groups of features used in our experiments. While some groups intrinsically contain temporal information (e.g., user history), others are independent of time. In this section, we study the relationship between the helpfulness of reviews and time.

For each Amazon review data in a given category, we obtain the date t_0 of the first written review. Then, for each review in the dataset t_i , we compute the quarter (of a year) number with $\lceil (t_i - t_0) / 90 \rceil$. Let us consider ‘Toys and Games dataset’, and the features number of tokens (or words) and star rating. Figure 4.4 shows that on average, older reviews tend to be shorter than newer ones. Similarly, older reviews tend to have a higher star rating. If we compute the Spearman correlation coefficient between the average number of words in a quarter and the quarter number, we find a positive correlation 0.55. Similarly, the star rating is negatively correlated -0.85 .

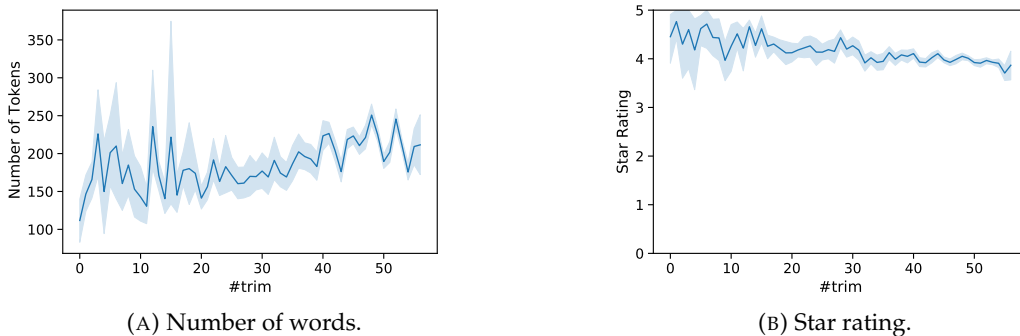


FIGURE 4.4: Temporal analyses on Toys and Game dataset.

We extended this kind of analysis to all features and dataset combinations. Figure 4.5 summarizes our findings via a boxen plot. In particular, for each feature category, we report its Spearman correlation distribution. Ideally, a family is ‘temporally independent’ if its distribution is dense toward 0. We can observe that review emotion, review structure, review metadata, product metadata, and reviewer history have median values close to 1 (or -1). Instead, review readability and embedding show a more uniform distribution, with a median value close to 0. These two feature families seem temporal invariant.

Our findings suggest that we are faced with a case of *concept drift* [295, 96]. Reviews stylistically changed over time, which might be natural if we think about reviews written in 2000 compared to those written in 2010. Furthermore, in its early stage, Amazon was used by a small subset of the world population. By 2010 on the other hand, Amazon’s popularity has grown exponentially. The number of Amazon Prime members went from 1 million in 2008 to 31 million in 2014 [235].

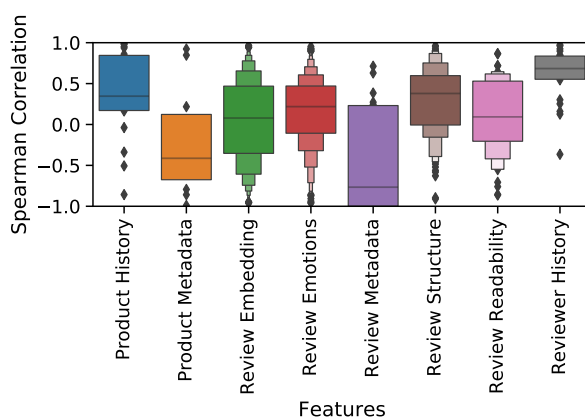


FIGURE 4.5: Distribution of the Spearman Correlation between different feature categories and quarter number. For each distribution, we further report the median value.

4.5 Classification Results

The goal of this section is to describe the efficacy of different features in predicting helpfulness. Section 4.5.1 describes our experimental methodology, followed by classification results in Section 4.5.2. Section 4.5.3 discusses how models generalize among different Amazon categories. We conclude with Section 4.5.4 by analyzing the models’ performance at the varying number of voters in the definition of the ground truth (see Equation 4.3).

4.5.1 Experimental Methodology

We used five well-known supervised classifiers in Scikit-Learn [40]: Logistic Regression, Decision Tree, Random Forest, AdaBoost, and Gaussian Naive Bayes. For each dataset, we trained these five models with the nice feature families introduced in Section 4.3.1 (i.e., eight feature families plus one that combines all of them), plus a case where all feature types are considered together. The training phase produced 270 models, i.e., 5 models times 6 datasets times 9 feature categories. We applied a grid search for each model to find an optimal (highest weighted F1-score) configuration thorough evaluation of the validation set. Note that the test set was blind

when these hyper-parameters were learned using the training and validation sets. The only preprocessing used was the StandardScaler, where each feature is standardized.⁵ We now briefly describe the parameters we search.

- Logistic Regression: $C = [0.01, 0.1, 1, 10, 10]$, $fit\ intercept = (True, False)$, and $class\ weight = [None, "balanced"]$.
- Decision Tree: $criterion = ["gini", "entropy"]$, $max\ depth = [None, 2, 3, 4, 5, 10]$, $class\ weight = [None, "balanced"]$, $min\ samples\ split = [2, 3, 5]$, $samples\ leaf = [2, 3, 4, 5]$.
- Random Forest: we use the same search of the decision tree, with the addition of $n\ estimators = [4, 16, 32, 64, 128, 256]$.
- AdaBoost: $n\ estimators = [2, 8, 16, 32, 64]$, $learning\ rate = [0.1, 1., 10]$.
- Gaussian Naive Bayes: $var\ smoothing = [1e - 11, 1e - 10, 1e - 9, 1e - 8, 1e - 7]$.

4.5.2 Classification Results

We now report the results of our experiments in Figure 4.6. We show a heat-map presenting the weighted F1-score of the combination of feature families and models for each dataset. We can immediately see a wide discrepancy between the F1-scores of different datasets: for example, the ‘Cell Phones and Accessories’ highest score has an F1-score of 63, while ‘Digital Music’ gets an F1-score of 73. These results suggest that the underlying mechanism linking a review with its utility score might differ based on the product type. We further note that classical features derivable from the review text (i.e., review readability, review structure) do not perform as well as other feature families (e.g, product history). This highlights the importance of considering the whole URI for building an optimal classifier.

We now study the impact of families of features and models in predicting helpfulness. To do so, we average the performance of the various families of features and models among the different combinations (i.e. of datasets, models, and feature families). The results are shown in Figure 4.7. Figure 4.7a shows that on average, models using all features have higher performance compared to those considering specific families of features. We see that features involving review metadata and product history show the best performance, only slightly below the case when all feature families are used. In contrast, product metadata features seem to be poorly connected with the helpfulness class.

Figure 4.7b shows that there is no real ‘winner’ amongst the 5 predictive models, i.e., the five models, on average, show similar performance. These results suggest that the model choice is not particularly relevant. Hence, the focus of future works might be on defining expressive features.

4.5.3 Cross-Domain Results

An interesting question raised in previous works is understanding the transferability of a helpful review classifier [317]. In other words, can a model trained in a specific domain (e.g., ‘Digital Music’) be used for a distinct one (e.g., ‘Electronics’)? This research question seeks to understand if there is a common underlying mechanism that relates reviews to their helpfulness between different product categories.

⁵scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

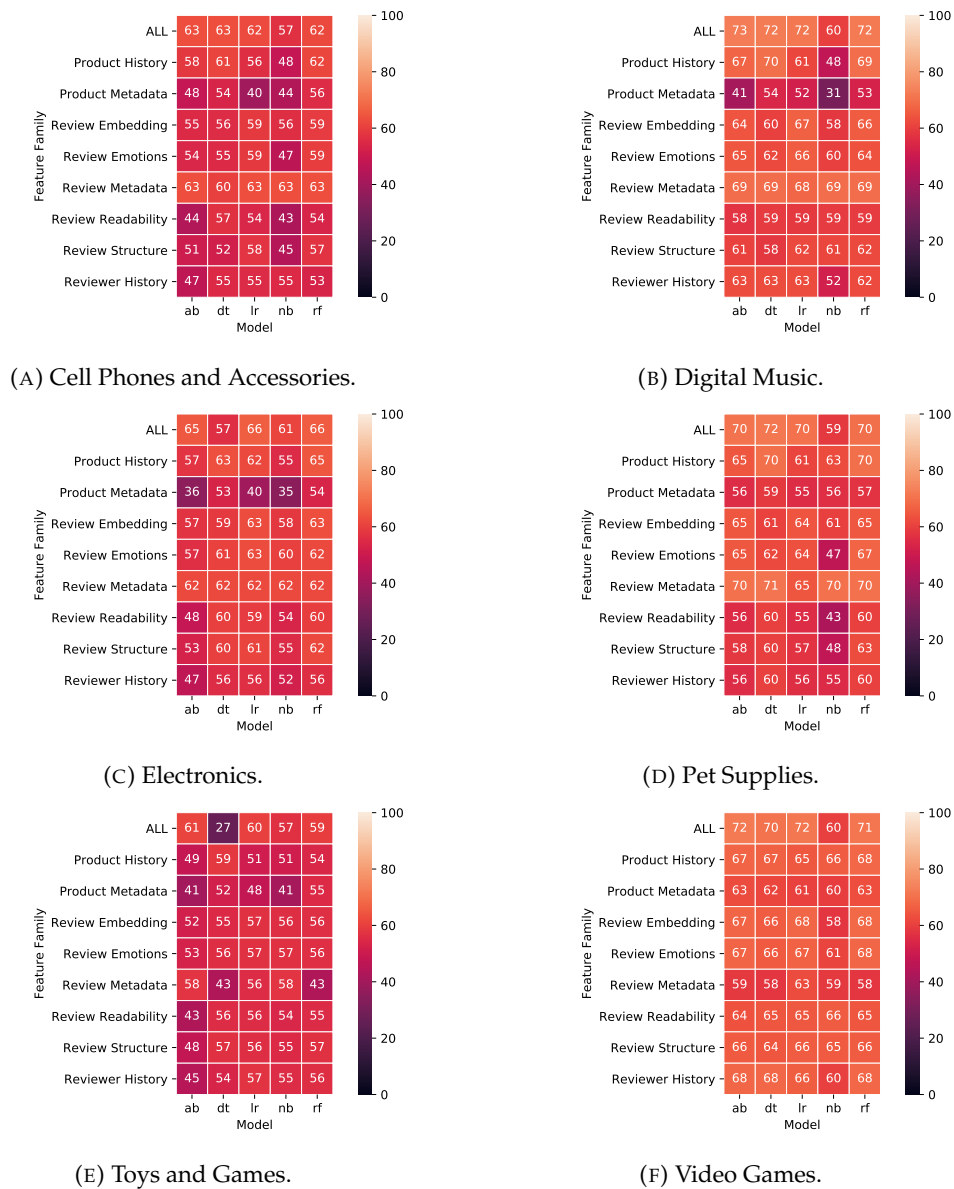


FIGURE 4.6: Weighted F1-score (the higher, the better) among different datasets.

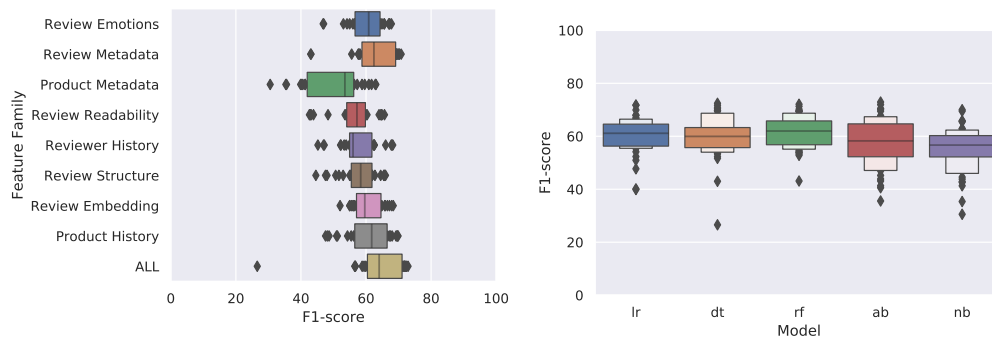


FIGURE 4.7: Average impact of features families (4.7a) and models (4.7b).

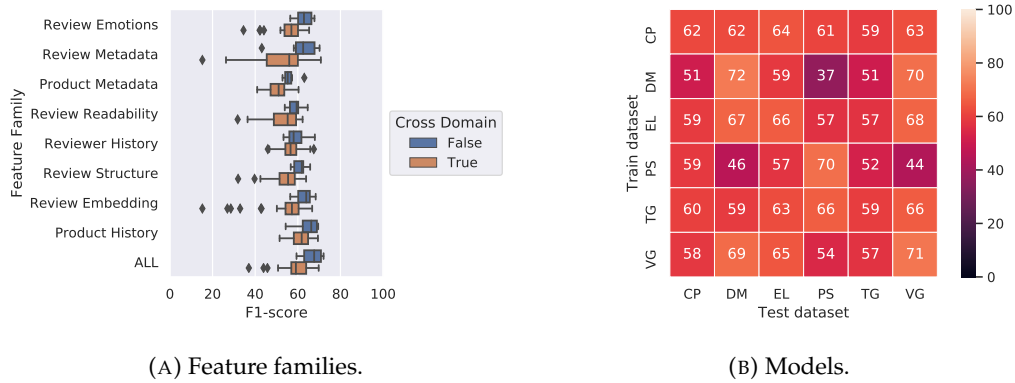


FIGURE 4.8: Cross dataset analysis of Random Forest-based models. On the left, the average F1-score at features level. In this graph, the models trained and tested on the same dataset are in blue, while cross-domain cases are shown in orange. On the right, the heat-map of models using ‘ALL’ features. The dataset name is reported with a compressed format. Starting from the left, Cell Phones and Accessories (CP), Digital Music (DM), Electronics (EL), Pet Supplies (PS), Toys and Games (TG), and Video Games (VG).

As Section 4.5.2 shows that all 5 predictive models have similar performance, we focused only on Random Forest-based models in this experiment. Another reason for this choice was that the Random had the lowest standard deviation of the 5 classifiers. Figure 4.8a shows the average F1-score produced by Random Forests when used with different families of features. First, we see that cross-domain performance is lower (as expected) compared to those where the test set belongs to the same category as the training data. Furthermore, some feature families seem to be strongly related to their original domain; for example, review metadata and review readabilities tend to have higher variance in their performance.

Finally, we focus on understanding the transferability among different datasets. We consider Random Forest-based models trained over all the feature families. Figure 4.8b shows the performance of the cross-comparison. The diagonal represents classifier performance when tested on the same category on which it was trained. We first see that there are only a few cases with poor transferability, e.g., ‘Digital Music’ and ‘Pet Supplies’ cases. Similarly, models tested on ‘Pet Supplies’ and, vice-versa, the model trained on ‘Pet Supplies’ and test on the other categories have poor transferability. On the other hand, several models tend to be consistent in their performance: for example, ‘Cell Phones and Accessories’ model shows similar performance among different datasets.

4.5.4 The impact of the ground-truth

We derived the ground truth using the number of positive votes and the total number of votes (see Equation 4.3) provided in the Amazon dataset. To have reliable ground truth, most works, including ours, discard samples that do not have at least N votes. In our case, we set $N = 3$. Nevertheless, this labeling process allows having ground truth with different weights of credibility. For instance, a helpful ratio of 0.75 may be derived from 3 positive votes out of 4, or with 30 positive votes out of 40. Naturally, the review with 40 votes is intuitive ‘more credible’, and our model should better learn to predict such an instance rather than the one with 4 votes. A straightforward solution is to increase N (i.e., the threshold), from three to 10, for

example. However, this approach would reduce the number of data samples. In fact, the Amazon review dataset presents a long-tailed distribution of the number of votes, e.g., Figure 4.9 shows the distribution of the number of votes in the Toys and Games category.

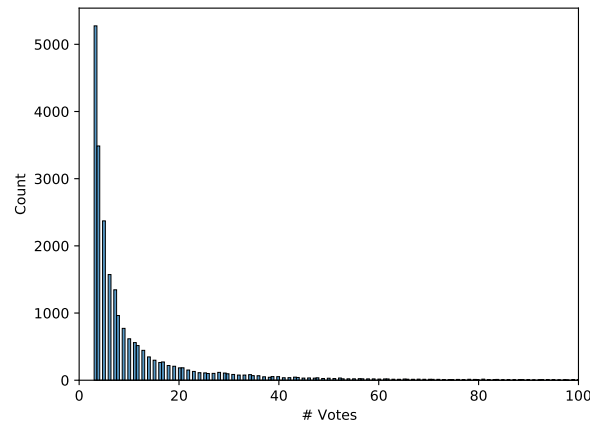


FIGURE 4.9: Long-tailed distribution of the ‘number of votes’ feature for the ‘Toys and Games’ dataset.

To understand if the number of votes has an impact in our models, we measure the performance of our models at different level of voters: *low credibility*, if the number of voters is in between three and five, *medium credibility*, if the number of voters is between 6 and 10, and *high credibility*, if the number of voters is greater than 10. In this phase, we tested only Random Forests considering all extracted features.

We report the results in Figure 4.10. Generally, reviews with a high number of votes tend to have a higher F1-score compared to those with fewer votes. Compared to low credibility reviews, on average, the improvement in F1-score is +6.97 when considering medium credibility reviews and +14.03 pp when considering high credibility reviews. Such discrepancies are significant and suggest that reviews with a small number of voters might be wrongly labeled. This phenomenon, combined with the long-tail distribution, might hide the true quality of classifiers. This finding suggests that future works must carefully consider the definition of ground truth.

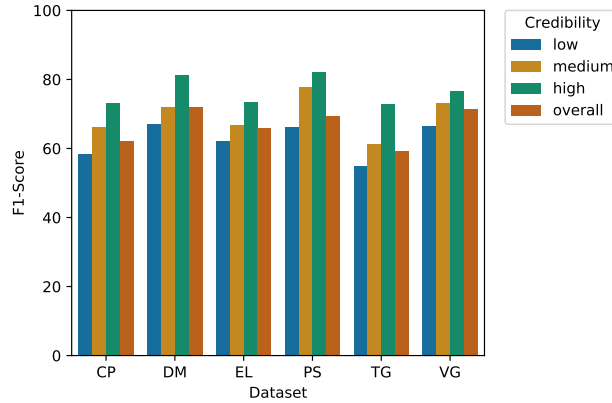


FIGURE 4.10: F1-score when varying of the ground-truth credibility. Starting from the left, Cell Phones and Accessories (CP), Digital Music (DM), Electronics (EL), Pet Supplies (PS), Toys and Games (TG), and Video Games (VG).

4.6 Deep Neural Networks for Helpfulness Classification

In the last 10 years, Deep Neural Networks (DNN) have been widely applied in many domains ranging from healthcare [87] to autonomous vehicles [114]. In this section, we analyze whether the helpful review classification task might benefit from recent techniques proposed in NLP. Section 4.6.1 presents the experimental settings. Section 4.6.2 presents the results.

4.6.1 Experimental Settings

Our experiments use the datasets presented in Section 4.3.2 and compare three types of models: *baselines*, i.e., the best performing models from Section 4.5, *fine-tuned*, i.e., pre-trained models that are fine-tuned for the helpfulness prediction task, *graph neural networks*, i.e., a DNN which is modeled leveraging the URI representation. Following the outcomes discussed in Section 4.5.4, for each model we report the weighted F1-score among the 4 dataset categories: low credibility, medium credibility, high credibility, and overall. We now briefly describe the experimental settings of the three models we test.

Baselines We consider the top three type of classifiers presented in Section 4.5.1: Adaboost (AB), Logistic Regression (LR), and Random Forest (RF). We use all features as that led to the best performance.

Deep Neural Networks The Review Platform Graph (URI) structure can be modeled with Graph Neural Networks (GNN). We are not the first to apply GNNs to this problem. Qu et al. [238] used GNNs to better model the impact of product categories in classifying helpfulness. In our implementation, we transform each dataset into a unique graph, where the vertices are the reviews. Two reviews v_i and v_j are linked if the two reviews are written by the same author or if they both belong to the same product. Each review is represented as an embedding containing the features we presented in Section 4.3.1. We model the task as semi-supervised learning, where at training time, only part of the nodes ground-truth is known. The model is

implemented using Torch-Geometric⁶. We used a grid-search to find the number of Graph Convolutional Networks (1,2), number of linear layers (0,1,2), and hidden dimension (32,64,128,256,512). An early stopping mechanism is set with patience equal to 5 epochs. Each model is trained for a maximum of 200 epochs.

Secondly, a popular research strand involves the use of *transfer learning* and *fine-tuning*. In the first, the knowledge acquired for a problem is used to solve a similar task. In the second, a model pre-trained for a problem is further trained with a very low learning rate for the target task. We focused on fine-tuning, which has previously shown strong performance in several NLP tasks [129]. We studied whether we can leverage state-of-the-art techniques to predict the helpfulness score by using the review text only. We use HuggingFace’s [310] implementation of DistilBert (DB) [254]. For each dataset, we fine-tuned the model for a maximum number of 10 epochs and applied an early stopping mechanism with patience equal to 3 steps.

Finally, we include a MultiLayer Perceptron (MLP) classifier, as done by Lee et al. [168]. Our implementation uses the MLP deployed by Scikit-Learn. Parameter selection is conducted over the number of hidden layers [(905,), (905, 452), (452,)], activation functions [*tanh*, *relu*], alpha [0.0001, 0.05], and learning rate scheduler [*constant*, *adaptive*].

4.6.2 Classification Results

Table 4.3 presents the results of the tested classifiers for the different datasets. For each dataset, we also analyzed the performance at different credibility levels, accordingly to the methods in Section 4.5.4. The best weighted F1-scores are highlighted. Generally, we see that baseline models outperform DNN models in 21/24 cases. DNNs present an outstanding performance in 3 of 24 cases. In general, AdaBoost and Logistic Regression obtain the best results. These results suggest that future works should pose their main focus on feature extraction and problem modeling rather than model selection.

⁶<https://pytorch-geometric.readthedocs.io/>

TABLE 4.3: F1-scores with varying the ground-truth credibility on the following Amazon categories: Cell Phones and Accessories (CP), Digital Music (DM), Electronics (EL), Pet Supplies (PS), Toys and Games (TG), and Video Games (VG). This table contains the performance of the following classifiers: Adaboost (AB), Logistic Regression (LR), Random Forest (RF), Graph Neural Network (GNN), DistilBert (DB), and MultiLayer Perceptron (MLP). Highlighted, the best F1 scores.

| Dataset | Baseline | | | DNN | | | |
|---------|----------|------|------|------|------|------|------|
| | AB | LR | RF | GNN | DB | MLP | |
| CP | low | 59.3 | 59.3 | 58.3 | 58.0 | 57.8 | 59.1 |
| | medium | 67.3 | 64.5 | 66.3 | 63.6 | 65.1 | 61.6 |
| | high | 73.8 | 72.1 | 73.1 | 76.3 | 74.5 | 69.9 |
| | overall | 63.0 | 62.2 | 62.0 | 61.8 | 61.3 | 61.0 |
| DM | low | 67.5 | 66.5 | 67.1 | 66.6 | 66.5 | 65.4 |
| | medium | 73.5 | 71.3 | 72.1 | 70.4 | 70.0 | 71.0 |
| | high | 82.5 | 81.9 | 81.3 | 81.4 | 80.4 | 79.5 |
| | overall | 72.9 | 71.7 | 72.1 | 71.4 | 71.0 | 70.4 |
| EL | low | 61.2 | 62.2 | 62.3 | 61.3 | 61.3 | 60.6 |
| | medium | 65.5 | 67.5 | 66.8 | 66.4 | 64.9 | 64.3 |
| | high | 74.9 | 75.9 | 73.4 | 74.7 | 72.3 | 71.1 |
| | overall | 65.2 | 66.4 | 65.8 | 65.4 | 64.5 | 63.6 |
| PS | low | 66.6 | 66.3 | 66.1 | 66.3 | 66.6 | 66.2 |
| | medium | 79.1 | 80.0 | 77.8 | 77.5 | 78.0 | 78.7 |
| | high | 85.1 | 80.2 | 82.1 | 76.0 | 83.9 | 81.7 |
| | overall | 70.5 | 69.9 | 69.5 | 69.0 | 70.1 | 69.7 |
| TG | low | 57.0 | 55.7 | 55.0 | 54.1 | 54.2 | 55.6 |
| | medium | 62.7 | 61.7 | 61.1 | 61.1 | 62.8 | 61.8 |
| | high | 73.1 | 75.7 | 73.0 | 72.4 | 72.0 | 69.9 |
| | overall | 61.0 | 60.4 | 59.4 | 58.7 | 59.1 | 59.2 |
| VG | low | 68.3 | 67.4 | 66.5 | 66.6 | 66.8 | 65.2 |
| | medium | 72.5 | 73.3 | 73.1 | 72.0 | 72.0 | 72.2 |
| | high | 76.7 | 77.0 | 76.6 | 76.3 | 75.4 | 73.4 |
| | overall | 71.9 | 72.0 | 71.3 | 70.9 | 70.7 | 69.5 |

4.7 Related Research Areas

Though this paper has already provided a comprehensive survey on work on predicting helpfulness of reviews, we briefly discuss two related bodies of work involve: (1) opinion spamming and (2) fake reviews generation.

Opinion Spamming Reviews are a powerful weapon that can affect the opinions of the crowd, making them an easy target for malicious actors and fraudulent behaviors.

Crowdturfing platforms are perfect for achieving such manipulations since they are related to illegal activities [209, 305]. For example, such actors can leverage crowdturfing campaigns to spread control reviews on target products.

Fake Review Generation Opinion spamming can be automated when combined with the power of generative language models, e.g., malicious applications that generate fake Yelp reviews [318, 146]. Yao et al. [318] showed that character-based LSTM artifacts are indistinguishable from real reviews and that human beings cannot spot such reviews. Furthermore, such sentences evaded fake reviews detectors proposed before their study that adopted supervised ML algorithms based on linguistic features and plagiarism checkers.

Building on top of [318], Juuti et al. [146] showed that the latter’s char-LSTM generator is hampered because they generate fake reviews without considering the target, e.g., the model generates a review for Italian restaurants while the target is Japanese cuisines. The authors thus proposed a Natural Machine Translation that encodes a list of targeted review’s properties in an n-dimensional vector (i.e., latent space) to overcome this issue. Then, a decoder generates a review containing such desired requirements.

4.8 Summary

This paper has organized the literature on helpful review classification with several significant contributions listed below:

1. The novel URI paradigm provides a single framework within which to consider the helpfulness prediction problem. Our experiments show that examining the whole review platform, not just the text of reviews, helps significantly improve the performance of algorithms to predict the helpfulness of reviews. The experiments presented in Section 4.5.2 show this conclusively.
2. The systematic review of past works in the area presented in Section 4.2 can better guide future researchers. The underlying mechanism that links a review to its helpfulness score is determined both by the intrinsic quality of the review quality as well as other contextual factors such as past reviews of the product, the type of product, and so on. The URI framework presented in this paper formalizes the combination of intrinsic review quality — and the context in which the review is written.
3. From a validation perspective, we found that strategies like random split and cross-validation might lead to erroneous results. Indeed, (3) helpfulness score analyses should adopt temporal splitting as a validation procedure. As shown in Figure 4.4, some features tend to change over time, meaning that reviews written ten years ago might be stylistically different compared to nowadays reviews. The notion of helpfulness review might be temporally affected as well. This outcome might inspire future research.
4. Only a few previous efforts offer repositories to study and replicate experiments. We invite future research to reverse this trend. Allowing reproducible baselines might allow future researchers to properly replicate and evaluate them, which is mandatory to identify the state of the art. The data, features and source code used in this paper will be made publicly available when this paper is published.
5. The classification results discussed in sections 4.5 and 4.6 suggest that the notion of helpfulness varies from one dataset category to another. What defines a helpful review in a video game might differ from a smartphone scenario.

While the search for a universal classifier might be valuable, it is equally valuable to improve the performance that we presented (and that can be found in previous works). The definition of new features for prediction might be just as important or more important than focusing on model selection.

6. The helpfulness prediction tasks hide distinct challenges that might limit proper understanding of the problem. As discussed in Section 4.4, properties that describe a review might change over time. Future researchers should thus focus on developing a solution that considers this aspect and proposes a model adaptable over time. Second, as discussed in Section 4.5.4, the credibility of ground truth might lead to error. It may be well worth it to explore unsupervised or semi-supervised learning techniques, limiting the impact of 'low credible' reviews.

Chapter 5

PRaNA: PRNU-based Technique to Tell Real and Deepfake Videos Apart

In the last decade, video content has become an enormous part of the Internet's data traffic. Videos are a handy form of communication, and as they increase their popularity, the number of contexts and ways in which they appear rapidly expands. Online news reports are increasingly delivered through video segments rather than articles, and politicians often make statements and communicate through live streams on their social media accounts. With the rise in popularity and availability, video content can now be used to hold individuals accountable for their words and actions.

The communication power of videos also attracted malicious activities such as revenge porn dissemination [276] and malicious video content manipulations [300]. In this work, we focus on this last domain application and, in particular, on deepfake videos detection. Deepfake videos rely on advanced techniques related to Deep Learning (DL) [292], like the Generative Adversarial Networks (GAN) [110] used to produce those artificial contents. Realistic face and voice swapping software are thus a potent and harmful tool in the wrong hands, as a well-placed manipulated video can easily attribute words, actions, or even just location to targeted/unaware people. Usually, deepfake generation processes require the victim face and a target video (see Section 5.1.2), where the nature of the video itself could negatively influence victims' lifestyles (e.g., public shaming, career destruction, and crime implications). For example, Figure 5.1 shows a frame taken from a deepfake video, where the authors of the clip control the speech of Barack Obama [300]. Target videos can be retrieved from various web sources, from blogs and social networks to video channels like YouTube.

The detection of such manipulation is thus fundamental, raising the interest of both research and industrial areas demonstrated by the Kaggle Deepfake Detection Challenge [80]. The current state of the art of deepfake detection methods is formulated as a binary classification problem and it is divided into two families, as reported by the surveys [292, 298] based on: i) *Hand-crafted features*, where the detection relies on the analysis of properties that can distinguish deepfakes from real videos (e.g., anomalous eye blinking [172], heart rate variations [60]), and ii) *DL-based features*, where the detection relies on analyses of properties extracted by DL-based architectures [214, 173].

While DL-based methods show promising results, they also show limited ability to generalize, i.e., difficulty to be robust on unseen data [152]. Moreover, the training procedure involves a non-trivial amount of data. In contrast, not much attention

has been put on identifying real videos among deepfakes counterparts (more than one). We believe that this trend (i.e., binary classification) is the result of the task demand, where usually we aim to analyze a single video. Vice versa, in real life we may think it is unlikely to have a pool of deepfakes originated by the same source and the source itself but it is not hard to find such artificial pool. For example, popular deepfake detection challenges like CelebDF [174] and DFDC [80] defined their task in this way, i.e., one benign video has one or multiple deepfake counterparts. Nevertheless, analyses of artificially created deepfake pool might uncover peculiar deepfake behaviours, and the findings might lead to increasingly more robust deepfake detectors.

Contributions In this paper, we address the deepfake detection task from the point of view of multiple-instances by leveraging on hand-crafted features from a set of videos that share the same source. In particular, we use the Photo Response Non-Uniformity (PRNU), a type of pattern noise that is present and unique in all the images acquired by a certain digital imaging sensor (e.g., a specific camera or smartphone). The first deepfake video detection strategy leveraging PRNU has been proposed in [159], where the mean normalized cross correlation is shown to be effective to discriminate real videos from deep fakes. However, different studies showed that PRNU being ineffective on more accurate deepfake video techniques [76, 188].

In this paper, we observe that classical PRNU features (e.g., normalized cross correlation) are not effective in a binary task, in agreement with [76, 188]. However, when similar analyses are applied to a pool of deepfake, we found a unique feature able to distinguish deepfake from their real counterparts. Using this finding, we design PRaNA, an algorithm to discriminate fake videos from their real version with an high accuracy.

We can summarize our contributions in two major points:

- We propose PRaNA, a new strategy able to discriminates real videos from deepfakes counterparts that share the same video source.
- An empirical evaluation of PRaNA among three datasets is given: the Deepfake Detection Challenge Dataset [80], the Celeb-DF [174], and FaceForensics-Deep Fake Detection [248].

5.0.1 Organization

The paper is organized as follows. In Section 5.1 we introduce basic concepts of PRNU, deepfakes generation and detection. In Section 5.2 we describe the datasets used in our analyses. We then describe and evaluate PRaNA, in Section 5.3 and Section 5.4, respectively. We conclude with Section 5.5.



FIGURE 5.1: A frame related to a deepfake video of Barack Obama [300].

5.1 Background & Related Works

In this section we introduce the Photo Response Non-Uniformity noise (Section 5.1.1), followed by the deepfake generation procedures (Section 5.1.2), and the deepfake detection methods in Section 5.1.3.

5.1.1 Photo Response Non-Uniformity

Photo Response Non-Uniformity (PRNU) is a pattern noise present on all photos taken by digital cameras. The nature of the pattern is given by different light sensitivities in each sensor of a camera's sensor array. The sensors' manufacturing process causes these variations, and the resulting pattern is unique to each device. PRNU is applied in image and video forensic in several applications, such as source identification and various manipulation detection types [189, 53]. We now describe the PRNU extraction process, which involves N images of a target camera.

The estimation process involves modeling the noise components of a photo. PRNU is multiplicative to the intensity of the captured light as it is a result of the altered sensitivity of imaging sensors. We can represent the final image I as

$$I = I^{(0)} + I^{(0)}K + \Theta, \quad (5.1)$$

where $I^{(0)}$ is the natural light signal coming into the camera, K the PRNU, and Θ is an additive noise signal composed by other noise sources (e.g., random shot noise, dark current noise). To deal with different noises, a denoising filter is applied. The final estimated PRNU \hat{K} is obtained by the residuals W through maximum likelihood:

$$\hat{K} = \frac{\sum_{k=1}^N W_k I_k}{\sum_{k=1}^N (I_k)^2}. \quad (5.2)$$

From now on we refer to K as the estimated PRNU component \hat{K} . At test time we can measure the amount of correlation between the noise residual of an image with K using different metrics such as the *Normalized Cross Correlation* (NCC), and the *Peak to Cross Energy* (PCE), where the higher the measure, the higher the correlation [106].

On videos, the PRNU is estimated on extracted frames from the media. Such estimation is more challenging on videos respect to images due to encoding processes that compress the resource, which affects the noise components. In video content,

we can find three different types of frames: *Intra* frames (I-type), i.e., frames independently coded that maintain a high amount of data, and *Predictive* frames (P-type) or *Bidirectionally predictive* frames (B-type), i.e., frames compressed with motion vectors obtained from their referent I-frame. The choice of the type of frames for the fingerprint calculation has an impact on the PRNU estimation process [286]. Even though videos shows challenges, in [162], the authors show that PRNU estimation is feasible even on heavily compressed YouTube Videos as long as the predictive blocks have a nonzero motion compensation and present their enhancement to estimation in that context.

5.1.2 Deepfake Generation

Following the notation introduced in [292], deepfake videos are divided into four families: *Entire Face Synthesis*, *Identity Swap*, *Attribute Manipulation* and *Expression Swap*. We focus on the identity swap because of its popularity and potential for harm. Identity swap techniques are divided into two generations of methods [174], based on their performance and visual achievements. While the *first generation* produces easy to spot videos due to coarse inconsistencies, the *second* one is much more challenging to detect (for humans and machines).

We now briefly describe the deepfake generation process. The reader might refer to [174] for more details. In the deepfake generation process, we have a *target* video (or origin) with a subject used as a base and a *donor* video with the victim's face. The goal is to transfer the victim's face of the donor video to the target one. Standard deepfakes techniques involve two autoencoders that learn to reproduce both the target and donor's faces. Here, the encoder E projects faces in the latent space, and the output feeds one of the two independent decoders: D_{src} and D_{trg} . As E is shared between the decoders and both are trained simultaneously, it is forced to learn to create a code representation of the faces that are identity-independent, capturing more abstract features about faces, like facial expressions. At test time, the target's latent representation is fed to D_{src} , resulting in a face substitution. Generally, both encoder E and decoder D are Convolutional Neural Networks (CNNs).

Another popular direction of deepfake artifacts generation is based on Generative Adversarial Networks [110]. For example, in [149] the authors present StyleGAN, a generative model to produce high-quality images (e.g., realistic faces).

5.1.3 Deepfake Detection

The current state-of-the-art in deepfake detection follows two main approaches [292, 298]: *learning-based* methods or *handcrafted features* techniques.

Deep learning is a popular strategy for many forensic solutions [9, 226], and it offers many solutions for the deepfake detection task [195, 248]. Many different approaches and improvements are proposed, such as attention map generation [71], recurrent layers to include the time dimension [119], optical flow analysis [43], and face regions separation [292]. Unfortunately, in general learning-based approaches show a lack of generalization to unseen manipulation types, with a significant drop in performance [152]. The aftermath is that identifying generalizable techniques is one of the primary directions that researchers are working on [315, 170].

The handcrafted-based features family focuses on specific features and properties of fake videos. The main advantage of such approaches is the high explainability. The downside is that they require more knowledge of the flaws of generation algorithms, and that they are made obsolete when specific countermeasures are

taken [226]. Some examples include the detection of biological signals like heart-beat [60] or specific visual artifacts [197]. Deepfake detection methods based on PRNU analysis also fall into this category. In [242] is shown the usage of PRNU to identify facial beautification. [159] showed that extracting the PRNU fingerprint of the face area from different sections of a deepfake video and correlating them against each other leads to lower correlations than real videos. However, in [76, 188], an opposite trend is found (i.e., the correlation is ineffective). In [188], the authors proposed a PRNU-based detector leveraging on prior works [258, 242] intuitions, and proposed PRNU spatial and spectral features to identify deepfakes.

To the best of our knowledge, past works like the ones presented in this section consider the detection as a binary task, i.e., understand if a video is real or deepfake. In contrast, our work face the deepfake identification problem from a multi-instances perspective, i.e., when multiple deepfake videos originated by the same source are found. To the best of our knowledge, we are the first to consider such scenario.

5.2 Datasets

In this work, we conduct our experiments on three second-generation (see Section 5.1.2) deepfake datasets: *Deepfake Detection Challenge Datasets (DFDC)*, *Celeb-DF*, and *FaceForensics-Deep Fake Detection (FF-DFD)*. These datasets provides pool of videos with one real and the others deepfake counterparts.

- *DFDC* was released in late 2019 as part of a challenge [80] [260], and was opened to wider research in 2020, after the competition. It consists of 124000 video clips, with the real footage filmed by a diverse set of actors in various conditions and deepfaked with eight different techniques.
- *Celeb-DF* [174] consists of 590 video clips from various celebrity interviews, and 5639 fake videos made by swapping identities among the real set with an improved deepfake algorithm made by the authors.
- *FF-DFD* [248], The dataset contains 3068 fake videos, and the data comes with three quality options: i) *RAW*, with high quality for both real and fake videos, ii) *c23*, with mid quality videos and iii) *c40*, with low quality videos. We used the *c23* option for our evaluation since it is the default compression level in many video encoding software packages.

These three dataset are suitable for our task since they contain several deepfakes versions starting from an original given video.

5.3 PRaNA

In this section we present “PRaNA”, the proposed algorithm able to spot real videos among deepfake counterparts. In Section 5.3.1, we summarize the hypotheses of the experiment. In Section 5.3.2, we introduce an analysis of the autocorrelation effect on deepfake, followed by some considerations on the role of the recompression (Section 5.3.3). We then describe in details PRaNA verification steps (Section 5.3.4), and conclude with possible frame selection strategies (Section 5.3.5).

5.3.1 Experimental Hypotheses

Earlier in this work (Section 5.1), we described that, in general, state-of-the-art models the deepfake video detection problem as a binary task, i.e., tell whether a video is real or generated with a deepfake algorithm. However, as proposed in several deepfake detection challenges (see Section 5.2), from a single original video, multiple deepfakes are generated using different *donors*.

Can we thus retrieve helpful information to address the deepfake detection task from such a scenario? To the best of our knowledge, we are the first to ask this question. In this preliminary experimental work, we start from the following hypotheses.

Hypothesis 1. *A pool of videos is given, i.e., a set of videos originated by the same target.*

Hypothesis 2. *Every pool of videos contains the original and unaltered target video.*

With Hypothesis 1, we assume that at the time of the analysis, we already have a pool of videos generated from the same *target*. How to retrieve such pools is out of the scope of this paper. In this work, we are mainly interested if we can extract useful features from a pool of deepfakes to address the deepfake identification task.

With Hypothesis 2, we limited our investigation to those pools that contain the original and unaltered target videos. This hypothesis cannot be held in real life, i.e., we do not know a priori if a pool of videos contains the original and unaltered target video.

The two hypotheses limit PRaNA application in the theoretical investigation of deepfake behaviours. With these hypotheses, we set an upper bound (ideal scenario) of the quality of the meta-features that we can extract from *pool* of deepfakes. Anyway, we believe that our investigation can give valuable insights to future works, by motivating and showing the potentiality of analyses that considers pools of videos.

5.3.2 Autocorrelation Analysis

The intuition that drives our investigation is that the PRNU is altered during the deep fake generation procedure. First of all we introduce the autocorrelation process used to correlate a fingerprint with a reference frame of a video. Given a video F , it can be considered as a temporal sequence of frames $F = \{f_0, f_1, \dots, f_m\}$, where f_i is the frame at time i , and m is the total number of frames. We can define the training set Tr and test set Te such that

$$Tr \subseteq F, Te \subseteq F, Tr \cap Te = \emptyset. \quad (5.3)$$

Following the procedure introduced in Section 5.1.1, we can extract the fingerprint K of the video F using the frames in Tr . Then, given K and the frames in Te , we calculate a list of correlations:

$$\vec{y} = NCC(K, f_i), \text{ with } f_i \in Te. \quad (5.4)$$

where the NCC is the normalized cross-correlation. Finally, the autocorrelation is computed as the average of the correlations. Note that we can use any correlation metric like the PCE.

We conduct this experiment on the Deepfake Detection Challenge Dataset (DFDC), where we select 100 real videos, and for each of them a fake version of it, for a total of 200 videos. For each video, we select 32 temporally equally distant frames to generate the fingerprint. To compute the autocorrelations, we select 16

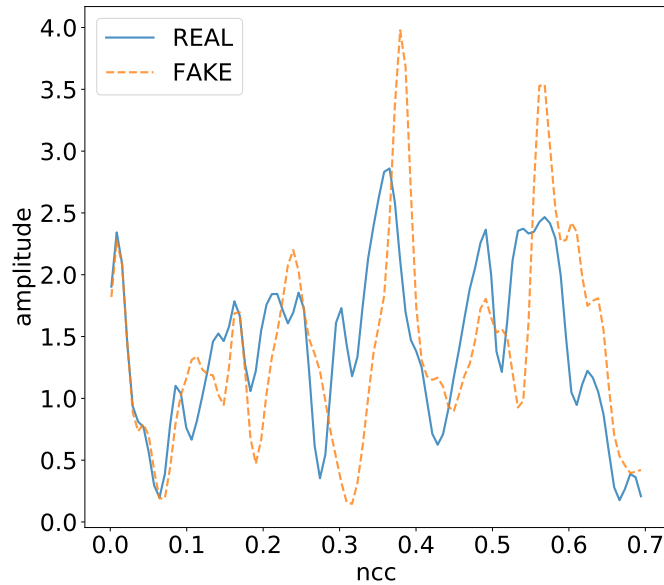


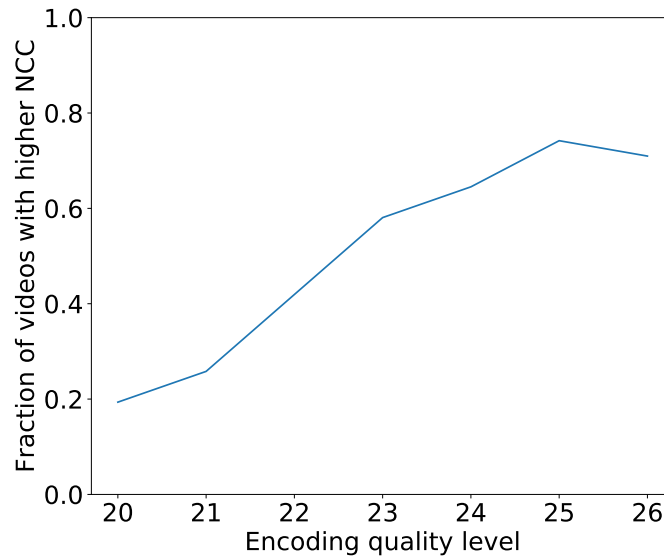
FIGURE 5.2: Distribution of autocorrelation for 100 real videos and 100 fake videos from the DFDC dataset, computed using NCC.

consecutively frames from the remaining set. In Figure 5.2 we show the comparison between the NCC distributions of real and deepfake videos. While the two distributions cannot be separated with a threshold value, we observe that when comparing the autocorrelation of a real video with its fake counterpart, the deepfakes have higher correlation for the 85% of the tested couples. This result suggests that autocorrelation is not suitable to perform deepfake detection in the absence of a counterpart sharing the same origin. However, this gives us the possibility to identify real videos among a group of deepfakes versions.

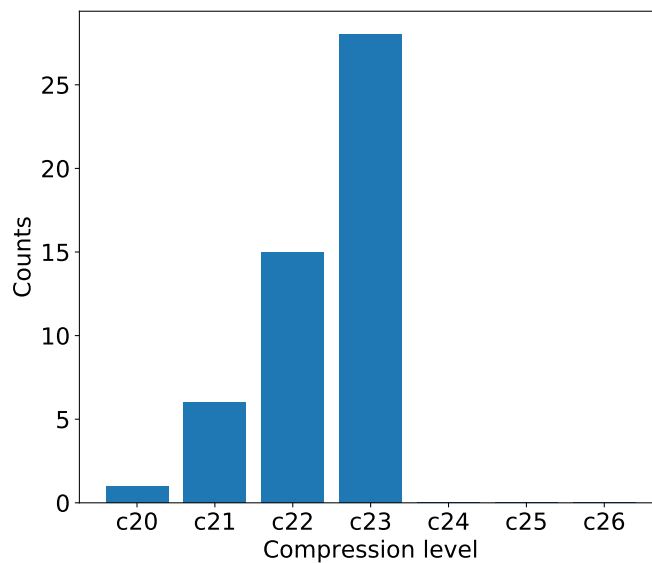
5.3.3 Considerations on recompression

Usually, videos are saved with lossy compression formats to reduce their sizes. During the deepfake generation process, an encoded video is manipulated and then re-encoded, and thus the PRNU might be affected from the recompression. We now want to understand whether the results presented in Section 5.3.2 are an effect of the re-encoding or of the deepfake creation. We measure this re-encoding effect on 50 random DFDC real videos using the x264 codec, the one used in the dataset and we vary our experiments on different compression factors, from $c20$ to $c26$. We measure how many re-encoded videos show a higher autocorrelation value after the re-encoding process than their original counterpart. Figure 5.3a shows an increasing trend, where the more we compress, the higher the fraction of videos with higher autocorrelation is.

To measure the real effect of re-encoding on our results, we now need to understand the encoding compression level of DFDC videos. To do so, we compare the video sizes of the 50 re-encoded videos on the range $[c20, c26]$ with 50 deepfake counterparts. Figure 5.3b shows the $c23$ seems to be the most likely compression level for DFDC videos, which is also the default compression level for 1920x1080



(A) Rates at which autocorrelation (using NCC) increases at various compression settings in DFDC videos.



(B) For each considered compression level, the number of fake videos whose size is the most similar to the re-encoded file size.

FIGURE 5.3: Recompression analysis.

x264 videos in many video encoding software packages. After guessing the compression settings (x264 at c23), we convert the 100 real videos used for the analysis described in Section 5.3.2, and we calculated their autocorrelations, comparing the results with the starting values of the original version. Only the 38% of the re-encoded videos show an increment in the autocorrelation value with respect to the original ones. We conclude that the phenomenon observed in Section 5.3.2 is not only the effect of the re-encoding, suggesting that the behavior we observed is a peculiar trace introduced in the PRNU by the process of deepfake generation.

5.3.4 Identification Algorithm

In Section 5.3.2 we show how deepfakes tend to have higher autocorrelations compared to their original versions. We used this concept to identify real videos when a pool of deepfakes counterparts is discovered. Formally, we define a *deepfake pool* P as set of N videos, where one is real and it is the source, and the rest $(N - 1)$ are the deepfake counterparts of four deepfakes using the real video as a source. The identification methodology is described in the Algorithm 1. The correlation metrics that we use are the NCC and the PCE, while the frame selection strategies are described in Section 5.3.5. The number of training and testing frames is parametric and limited by the videos' lengths.

Algorithm 1: PRANA

Input : N videos, frame selection strategy FS , correlation metric CM
Parameters: Number of training frames a , number of testing frames s
Output : Index of the real video

```

for  $i = 1, \dots, N$  do
  Load video  $V_i$  into memory as a list of frames for  $r \in FS_a(V_i)$  do
    Compute noise residual  $W_r$  for the frame  $V_i[r]$  and append it to
     $train\_set_i$ 
  end
  Estimate the fingerprint  $K_i$  of  $V_i$  from  $train\_set_i$  Create a list  $T_i$  of  $s$  test
  frame indexes from  $\neg FS_a(V_i)$  for  $e \in T_i$  do
    Compute noise residual  $W_e$  for the frame  $V_i[e]$  Compute the
    correlation  $CM(K_i, W_e)$  and append it to  $test\_results_i$ 
  end
  Compute the average correlation  $C_i$  from  $test\_results_i$  for video  $V_i$ 
end
return the index  $m$  of the minimum  $C_m$ 

```

5.3.5 Frame sampling strategies

The autocorrelation strategy described in Section 5.3.2 relies on the analysis of a single video. While the amount of available frames is potentially not very high, it is crucial to keep the two sets of frames disjointed, as the same frames in both sets can affect correlation measures. We now describe three possible frame selection strategies.

1. *First-N*. The approach uses the first n frames for Tr and the next s frames for Te . While easy to implement, it might suffer from the high similarity among frames, resulting in a PRNU that might contain also scene elements.
2. ii) *Equidistant*. The approach creates a subset of equally distant samples for Tr , while keeping the remaining frames for Te . This ensures more diverse scene contents across the training samples.
3. *I-P Frames Only*. The approach selects high-quality frames for Tr , i.e., starting from I-frames, and falling back to P-frames if the training number is not reached. This approach should guarantee higher quality frames for the PRNU estimation (see Section 5.1.1).

5.4 Results

In this section we show hyperparameter selection (Section 5.4.1), and results (Section 5.4.2) of PRaNA.

5.4.1 Hyperparameter Search

We optimize the parameters of our method with a grid search approach over the following ranges and options. i) *Number of training frames*. We use 16 to 80 frames with an increment of 16 for fingerprint estimation. ii) *Frame sampling method*. We use First-N, Equidistant and I/P Frames Only. iii) *Number of testing frames*. We use 16 and 32 testing frames. iv) *Correlation Metric*. We use NCC and PCE as metrics. PRNU estimation is performed with the `prnu-python` library¹, which also provides functions for the computation of NCC and PCE. Table 5.1 shows the results of the search using 50 groups, each composed of one real video and four fake versions of it (250 videos total), randomly selected from the DFDC dataset. The best configuration consists of an equidistant frame selection strategy, NCC as the metric, 64 training and 16 test frames.

TABLE 5.1: Accuracies for real video identification at various parameters on the DFDC dataset. In **Bold** the best accuracy. N/A for settings with not enough frames to run the task.

| Selection, Metric, Test | Number of training frames | | | | |
|-------------------------|---------------------------|-------|-------|--------------|-------|
| | 16 | 32 | 48 | 64 | 80 |
| First-N, NCC, 16 | 56.0% | 60.0% | 72.0% | 68.0% | 64.0% |
| First-N, PCE, 16 | 58.0% | 64.0% | 68.0% | 64.0% | 64.0% |
| First-N, NCC, 32 | 58.0% | 64.0% | 64.0% | 64.0% | 62.0% |
| First-N, PCE, 32 | 60.0% | 60.0% | 70.0% | 64.0% | 62.0% |
| Equid., NCC, 16 | 64.0% | 76.0% | 80.0% | 84.0% | 74.0% |
| Equid., PCE, 16 | 68.0% | 68.0% | 76.0% | 74.0% | 72.0% |
| Equid., NCC, 32 | 68.0% | 70.0% | 80.0% | 80.0% | 70.0% |
| Equid., PCE, 32 | 68.0% | 66.0% | 66.0% | 78.0% | 72.0% |
| I/P frames, NCC, 16 | 72.0% | 64.0% | 60.0% | N/A | N/A |
| I/P frames, PCE, 16 | 64.0% | 62.0% | 68.0% | N/A | N/A |
| I/P frames, NCC, 32 | 68.0% | 70.0% | N/A | N/A | N/A |
| I/P frames, PCE, 32 | 70.0% | 68.0% | N/A | N/A | N/A |

5.4.2 Evaluation

We evaluate our method with 150 groups of 5 videos for each dataset, using the best parameters shown in the previous section. Note that the 150 groups used for the evaluation of the DFDC dataset do not contain any group used for the hyperparameter search. Figure 5.4 shows the accuracy of our proposed methodology varying the number of deepfake counterparts. We can notice that our strategy outperforms the random guess in every configuration, with robust results on both DFDC and Celeb-DF datasets. Our results confirm PRaNA’s robustness and transferability, with accuracy up to 66% when considering one real video in a pool of four deepfakes using the real video as a source, and up to 80% when only one deepfake is considered.

¹github.com/polimi-ispl/prnu-python

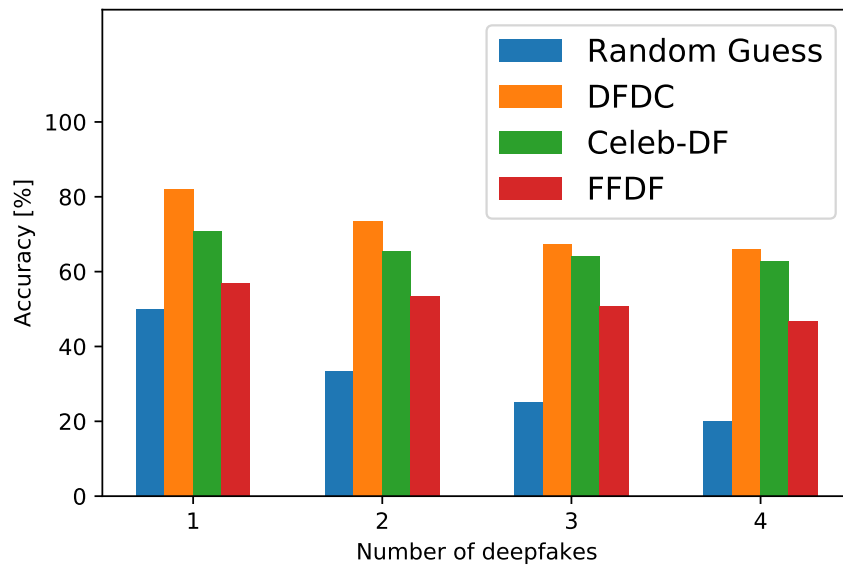


FIGURE 5.4: Accuracies at varying group sizes.

5.5 Summary

The spread of highly-realistic deepfakes threatens the trustworthiness of information that we reach online, from forums to social media. The identification of robust strategies capable of identifying deepfakes is thus fundamental. These strategies must show both robustness and transferability to face a wide range of deepfake generation techniques. Current detection strategies mainly determine whether a video is a deepfake. However, multiple-versions of the same video can be found, and forensic analyses of these set of video can be helpful in the deepfake identification task.

In this paper, we investigated the role of PRNU in such a scenario, and discovered that deepfakes tend to have higher autocorrelations compared to their real versions. Our findings can be applied as a first response in scenarios like social network video uploads, where multiple fake versions of the same video can be spread. Our proposed identification technique does not require any training and shows good robustness and transferability, capable of identifying different types of deepfakes among three well-known datasets.

While this work show promising results and research directions, it also present the following limitations that will be addressed in a near future.

1. PRaNA requires that a pool of videos is given (Hypothesis 1). This hypothesis might not be hold, since target and unaltered videos might not be publicly available. Moreover, even with the publicly existance of such videos, the retrieval of pool of deepfakes might be expensive and not a trivial operation to accomplish.
2. PRaNA requires the real-video (the target) to be present in the pool of deepfakes (Hypothesis 2). This represent a major limitation in many real-life scenarios, since it is unlikely to know a priori that the real video is present in a pool, while pools might contain only deepfakes.

3. Compression techniques applied in online platforms (e.g., Facebook, Youtube) can impact on the PRNU, and thus, PRANA might be affected. Future investigation should clarify such aspect.
4. We empirically identified that the PRNU autocorrelation tend to be higher in deepfake videos compared to their original sources. However, the reasons of this phenomenon must be clarify in future.

In conclusion, we believe that this work can inspire future research directions by looking at the problem from a multi-instance perspective. Comparing multiple deepfake videos generated from the same source can allow new hand-crafted feature comparisons and analyses, offering new tools to forensic analysts. Possible future directions might involve a deeper investigation of the PRNU autocorrelation behavior and how the deepfake generation process impacts such a metric. Similarly, future investigations might analyze how other well-known hand-crafted features (e.g., visual artifacts) behave when considering multiple deepfake videos sharing the same source.

Part III

Adversarial Machine Learning

Chapter 6

Introduction to Adversarial Machine Learning

This chapter depicts an overview of the Adversarial Machine Learning (AML) discipline. In depth discussion are presented in the following chapters 7 and 8.

6.1 Preliminaries

This section presents preliminary concepts to describe attacks [22, 132, 221].

Taxonomy Adversarial attacks can be describe using the following properties:

- *Influence* of the attacker to the victim's model. The influence can be *causative*, if the attacker alter the training procedure, or, on the opposite, *exploratory*, if the attacker's actions do not alter the victim's model.
- *Security Violation* level exploited by the attacker. An attack can target model's *integrity*, if the performance are negatively altered, model's *availability*, if the target model becomes unusable, and model's *privacy*, if the goal is to obtain sensible information (e.g., training data samples).
- *Specificity* of the attack. In the *targeted* mode, the adversary focus on a specific set of points to affect, while in the *untargeted* this constraint is relaxed, and the target is a more generic class of points.

Attacker's knowledge While the taxonomy describes attacks properties, we further need to integrate information about the attacker's knowledge [25]. Indeed, as in every cybersecurity threat scenario, the more the attacker knows about the victim's system, the stonger the attack. Generally speaking, state of the art define two distinct - and opposite - cases:

1. *Black-box* scenario, if the attackers has limited knowledge, making the attack more realistic;
2. *White-box* scenario, if the attacker has information about the target system (e.g., gradient, model's architecture, training data); this case can be seen as the worst-case scenario.

6.2 Threats Landscape

This section summarizes popular attacks in AML, following the survey prposed by Chakraborty et al. [47].

Evasion Evasion attacks is the most popular family of threats studied in the AML literature [22, 25]. In this scenario, the adversary modifies a target sample to force a wrong classification on the victim’s model. For example, in the cybersecurity context, an adversary may attempt to craft a spam email to elude spam filters, or to modify malware to produce false negatives in malware detectors. More formally, be x a sample, f the victim’s function, $y_i = f(x)$ the output of the classifier, the adversary aims to find a small perturbation ϵ s.t.:

$$f(x) \neq f(x + \epsilon). \quad (6.1)$$

The nature of ϵ depend on the domain in which x is defined. For example, if it is an image, the perturbation might be the result of a pixel modification. If it is a piece of text, the perturbation can be the result of a character modification, word synonym, or paraphrasing. Most literature of evasion attacks aim to define strategies to compute ϵ , which is, usually, obtained through an optimization process. For instance, the Fast Gradient Sign Attack (FGSM) leverages the gradient of $f(x)$ to modify the input sample aiming to minimize a certain loss, e.g., targeting $f(x) = y_j$. If x is an image, ϵ is a mask of pixels that, if added to x , result into a misclassification. Another example of attacks in computer vision is the One Pixel Attack [278]: here, the optimization process aims to identify only one pixel which, if modified, creates a successful evasion.

Poisoning If an attacker has the ability to interact with the training data, he/she can contaminate the training process to compromise victims’ model performance [30]. Thus, while evasion attack is generally executed at testing time, i.e., the attack occurs to an already deployed model, poisoning attack occurs during the model training. In poisoning attacks adversaries can adopt two strategies, that can be combined [47]:

1. *Label flipping*, where the attacker can modify the ground truth of a subset of training samples;
2. *Sample manipulation*, where the attacker can arbitrarily craft the content of a subset of training samples.

Poisoning attacks’ literature is wide. For example, Biggio et al. [29] showed that adversaries can strongly degrade SVM performance by just randomly flipping training labels. On the opposite, if attackers craft adversarial samples, they can shift models decision boundaries, which, as a consequence, alter models performance [157, 30]

Backdoor Poisoning attacks are not only meant to degrade model performance. *Backdoor attacks* (or *trojan*) aim to insert triggers in the victims model, while preserving the overall quality of the model. These triggers can be activated at testing time by the attacker, producing as a result an effect similar to the evasion. The idea of a trigger is straightforward: let’s consider the image domain, a trigger can be an artificial pixel pattern that is unique and in common in a small set of images belonging to the same label. Then, victims model learns during the training phase an association between such pattern and the label. At testing time, if the attacker insert such malicious patterns in a sample, the predicted class will be the one associated to the trigger. The idea of backdoor attack was originally proposed in BadNet [117]; here the attack strategy combines both *sample manipulation* to insert a pixel-based trigger and *label flipping*.

Model Inversion Attackers can exploit ML not only to provoke misclassification or deteriorate its performance, but to gain sensible information about the training data. The first attempts of model inversion was conducted in a task involving a linear regression aiming to predict the drug dosage given patients personal information (e.g., medical history, genetic markers). The attack aims to infer sensible information (e.g., genetic marker) of patients. Example of such attack are conducted in both white-box and black-box settings [94, 93].

Model Extraction Part of the exploratory attacks involve the model extraction, where the adversary's primary goal is to steal (or learn) an exact replica of the victim's model. The attack is usually executed through an interaction between the adversary and the victim's model, where the attacker forward a set of sample and uses models predictions to learn a *surrogate model*. An example of such attack is given by Tramér et al. [293], consisting in an *equation-solving* strategy.

Inference Attack Attackers can further gain information about the training samples with *inference attack*. In particular, as shown by Ateniese et al. [18], an attacker can leverage a meta-classifier to infer sensible information from a black-box model, such as understanding whether a sample is part of the training set. Similarly, Shokri et al. [264] proposed *membership attack* aiming to identify whether a sample belongs to the same distribution of the training sample.

6.3 Adversarial Transferability

Generating adversarial samples for a target model is often unpractical (black-box setting). Interacting with the victims' model is indeed unfeasible in most of real cases, and thus adversaries need to address this issue. Attacker can thus generate an adversarial sample with a *substitute* model f' and then attack the victims' model f . The *adversarial transferability* is the ability to generate successful transferable adversarial samples [225]. In this work, the authors show that adversarial samples can transfer across different types of machine learning models, e.g., an adversarial sample crafted on an SVM can evade also a neural network. The authors further propose a technique to reduce the number of queries required to create a surrogate model trained on a dataset labelled by the target architecture.

The relationship between the victim's and the substitute models can vary: if the two models use the same algorithm (e.g., both random forest) we refer to *intra-technique transferability*, otherwise if the algorithms differ (e.g., random forest and neural network) we refer to *cross-technique transferability*. Its mathematical formulation is proposed by Demontis et al. in [77], where the authors define a set of properties that a good surrogate model should have to generate adversarial samples that are likely to transfer. In particular, for evasion attacks, the attacker should decrease the complexity of the surrogate model by adjusting its hyperparameters.

As stated in [182], most of the state-of-the-art transferability studies focus on small datasets (e.g., MNIST, CFAIR10), which is a limitation for most real-world scenarios. Yanpei et al. in [182] thus conducted an extensive study of adversarial samples transferability on state-of-the-art computer vision models, i.e. ResNet, GoogLeNet and VGG-16, trained over ImageNet [250].

Chapter 7

Fall of Giants: How popular text-based MLaaS fall against a simple evasion attack

Without any doubt, machine learning applications had considerable success in the 2010s, finding space in different areas, from the automotive industry with autonomous vehicles [252] to the biomedical sector with brain tumor segmentation [121]. The popularity of machine learning (ML) had a boost-up thanks to the increase of machines' computational power, making ML easily accessible to researchers and industrial developers, one of the significant obstacles of previous decades. Even though ML nowadays is accessible to developers, we can find three main limitations in the deployment of ML solutions, due to the lack of: i) amount of data required to train a robust model, ii) amount of computational resources, and iii) machine-learning engineers with suitable expertise. For instance, we can consider the task of sentence language translation: in 2016, Google presented a translator based on Long Short-Term Memory (8 layers both encoder and decoder), trained over a parallel corpus¹ of 26 million sentences (English-French) [312]. Not only the difficulty of the model's architecture implementation (i.e., the choice of hyperparameters), but it requires an enormous amount of resources: the training procedure involved the use of 96 NVIDIA K80 GPU, with six days of computation.

The aftermath of such complexity is that real-world tasks are unlikely modeled with ML by companies or users without enough resources (i.e., computational power, data, ML engineers). To overcome this issue, the principal IT organizations (e.g., Amazon, IBM, Google, Microsoft) started developing solutions for common complex tasks (e.g., text analysis, optical character recognition) called Machine-Learning-as-a-Service (MLaaS), where users pay for a certain amount of queries. In this way, for example, companies that require to analyze documents can use advance and well-performing techniques at an affordable price without caring for the complex training process. MLaaS had a discrete success, and in 2019 this market was valued 1.0 billion USD, with an estimation of 8.48 billion USD by 2025 [138].

In this paper, we focus on the text-domain, where the addition of malicious perturbation is translated with the modification of text through various techniques (e.g., misspelling, typos, word addition). The primary constraint of attacks in the text-domain is the "readability preservation", i.e., a human being can understand the meaning of modified sentences. The reasons behind the limitation mentioned above are deducible: let us consider the sentiment classification of music reviews (i.e., positive/negative reviews), where the adversary goal is to have positive classifications

¹A parallel corpus is a dataset used for sequence-to-sequence tasks, where each sample has a source and a target. The goal of the model is to translate the source to the target.

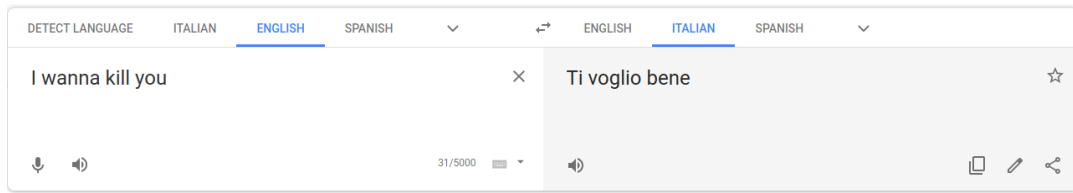


FIGURE 7.1: Zero-Width (ZeW) on a real-life scenario: Google Translate. The translated sentence means “I love you”.

for negative sentences. The original sentence could be “I hate this album”, and its malicious counterpart “I hA.XYztXaeX this album”. While the original sentence classification is expected to be negative, its counterpart might not; however, from a human point-of-view, the malicious sentence is not readable, and thus the adversarial sample loses its semantic meaning. So far, several works have proposed adversarial techniques in the text-domain, combining complex algorithms to find a trade-off between the effectiveness of the attack and the readability of produced malicious sentences. For instance, in [253], the authors identify the importance of the words of a given sentence for the target model, and replace them with sophisticated linguistic strategies. In [98], the authors proposed *DeepWordBug*, a process that first uses a scoring function to identify critical tokens for the target model and then applies character transformations to minimize the number of modifications. For an in-depth overview of state-of-the-art adversarial machine learning in text-domain, we suggest [325].

Contributions Motivated by the common assumption of “readability preservation”, we investigate a novel evasion technique that guarantees full readability and attack effectiveness. Our technique, called “Zero-Width” (ZeW), injects malicious UNICODE characters often used in text steganography strategies. These characters are called *zero-width space*, and their effect is that, when printed, they have zero-width, resulting invisible from a human being’s perspective. In one of our attack scenarios, we attack the popular web application Google Translate² on the English-Italian task. Figure 7.1 shows an example of wrong translation, where the original sentence “I wanna kill you” is translated as “ti voglio bene”, which means “I love you”. It is curious to notice that the input section has 31 characters (Figure 7.1, left side), while the sentence should contain only 16. In contrast to the state-of-the-art, ZeW does not require any assumption of the target model, and the readability constraint is relaxed. Moreover, so far, most of the proposed attack strategies aim to leverage the learning strategies’ weaknesses (e.g., model architectures); however, a ML application is composed of several stages (pipeline), where the ML model is only one of them. In this work, we aim to attack and disrupt the “indexing-stage” (see Section 7.1.1), which is the step that converts a sentence from the textual representation to a numerical one. To the best of our knowledge, not much attention has been put to find possible weaknesses to the entire text pipeline.

In this paper we aim to understand the following: i) the effect of ZeW attack on different types of indexing strategies, and ii) if commercial solutions are vulnerable to ZeW attacks. We conduct our experiments through a case study of a possible ZeW attack application: hate speech manipulation. We designed a simple injection strategy that, given a hateful sentence, identifies negative words and injects malicious characters in two possible fashions: i) *Mask1*, where only one malicious character is

²<https://translate.google.com>.

inserted in the middle of the word and ii) *Mask2*, where one malicious character is inserted between each character of the word. We tested this strategy over popular text MLaaS provided by Amazon, Google, IBM, and Microsoft, without having prior knowledge of the target models. The analysis aims to understand which services can be affected by ZeW, and the magnitude of the vulnerability. Our experiment shows that 11 out of 12 MLaaS are vulnerable to the proposed attack. We further introduce a simple countermeasure approach that can prevent ZeW. The purpose of this work is to emphasize the importance of studying the security of machine learning pipelines in all of their stages. Due to the gravity of ZeW, at the time of submission, all of the companies (i.e., Amazon, Google, IBM, and Microsoft) are informed.

Our contributions can be summarized as follows.

- We propose a novel text evasion strategy called *Zero-Width* (ZeW) that affects the indexing stage of text pipelines.
- We show the effect of ZeW over Machine-Learning-as-a-Service developed by Amazon, Google, IBM, and Microsoft. Out of 12 tested services, 11 show vulnerabilities (8 strongly affected).
- We propose a countermeasure to ZeW that can be easily integrated in every text ML-based pipeline.

Organization The chapter is organized as follows. In Section 7.1 we first briefly introduce the basic concepts required to fully understand the rest of the paper. Motivations, theoretical perspective, and countermeasure of ZeW are described in Section 7.2. We then move in Section 7.3 with the implementation of ZeW in a real case scenario, the hate speech manipulation, followed by a discussion of the attack results in a controlled environment first (Section 7.4), followed by MLaaS (Section 7.5). In Section 7.6 we summarize state-of-the-art attacks targeting models of MLaaS. We conclude with the limitations of the proposed attack in Section 7.7, followed by considerations and discussions of the possible implications of our results (Section 7.8).

7.1 Preliminaries

7.1.1 Text Pipeline

ML-based applications on text-domain follow a common pipeline, as described in [160, 325], and shown in Figure 7.2. The pipeline consists of the following compo-

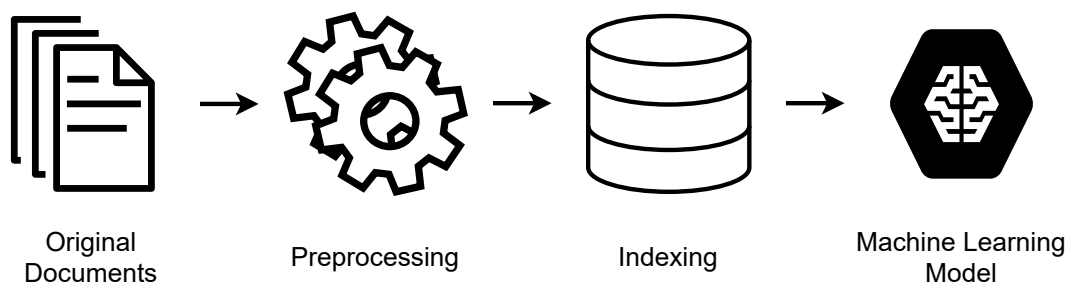


FIGURE 7.2: Machine Learning pipeline in Natural Language Processing.

nents: original documents, preprocessing, indexing, and machine learning model.

Original Documents Collection of the corpus of documents to analyze. The origin of these documents can differ, such as text files, PDFs, or HTML web pages.

Preprocessing Set of mechanisms that prettify documents, with the removal of useless information (e.g., TAG, format controls). This stage can involve different techniques, such as *tokenization*, where sentences are decomposed in lists of words, *stop-word removal*, where common words (and meaningless) are removed (e.g., articles), and *stemming*, where words are converted in their root form (e.g., books → book).

Indexing The mechanism that converts the symbolic representation of a document/sentence into a numerical vector. At training time, a vocabulary V of the possible representations (word/character level) is defined. The vectorial representation is usually handled in three possible ways:

- *Word-count encoding*. Each document is represented as a vector of words occurrences. For example, given the sentences $s_1 = \text{"hello there"}$ and $s_2 = \text{"hello hello"}$, a vocabulary $V = [\text{hello, there}]$, the sentences are represented as

$$\begin{aligned} s_1 &= [1, 1], \\ s_2 &= [2, 0], \end{aligned}$$

where the numbers represents the number of occurrences of the correspondent index in the vocabulary (i.e., "hello" in position 0, "there" in position 1). A variant of the word count often use the Term Frequency-Inverse Document Frequency (TF-IDF); this encoding tries to capture the importance of a word in the document given a collection of documents.

- *One-hot encoding*. This encoding represents a document as a list of vectors (one per word/char in the document). Given the previous example, the sentences are represented as

$$\begin{aligned} s_1 &= [[1, 0], [0, 1]], \\ s_2 &= [[1, 0], [1, 0]]. \end{aligned}$$

- *Dense encoding*. In this category, we find word embeddings, powerful vectorial representations of words [201, 202]. Here, each word is represented as a vector of real numbers (abstract representation). Dense representations can be pre-trained or trained end-to-end (e.g., using Language Models). For example, given the words "dog", "cat" and "hello", the dense representation indicates that the word "dog" is spatially closer to "cat" rather than "hello".

During the indexing phase, the pipeline needs to deal with unrecognized items (word/character level), i.e., items out-of-vocabulary (OOV). There are two possible ways to deal with it: i) discarding them from the analysis or ii) mapping them to a special token "UNK". In the latter case, the special token has a proper representation, based on the indexing strategy (i.e., word-counting encoding, one-hot encoding, dense encoding). The problem of unrecognized items is well-established in NLP, where the frequency of items in a dataset usually follows long-tail distributions. To reduce the complexity of the problem, the standard approach is to maintain small vocabularies with the most frequent items [55]. In Natural Machine Translation tasks this could be a problem, where all of the OOVs are mapped to a single token "UNK". For example, we can consider the translation task from English to

a target language of the sentence “Liam meets Noel”. Likely, both proper names are not present in the vocabulary, and thus they are mapped to the same token (i.e., “UNK meets UNK”), losing the name information in the target language. A standard approach, proposed in [190], consists of using placeholders to map rare items with unique pointers (e.g., “UNK1 meets UNK2”, where UNK1 = Liam and UNK2 = Noel) with a final name replacement in post-processing.

Machine Learning Model The ML model used for the task. The set of models vary from simple architectures (e.g., Logistic Regression, Random Forest), to Neural Networks (NN) and Deep Neural Networks (DNN). In the latter case, we can find variants of Recurrent Neural Networks (RNN) such as Long Short-Term Memory (LSTM) [123] or Gated Recurrent Units (GRU) [59].

ZeW aims to attack and disrupt the pipeline by affecting the indexing stage, affecting the ML-model performance.

7.1.2 Challenges of adversaries in Text-Domain

While AML gained popularity in Computer Vision (CV) from its early stages, only in recent years researchers moved onto the NLP domain. As identified by [325], three major aspects differentiate AML in NLP from CV.

- *Input Domain.* While images are defined in a continuous space (e.g., RGB matrix), sentences are discrete and represented as a list of symbols. It implies that the meaning of perturbation that we want to add changes its nature. For example, in CV the perturbation is defined as a matrix of values to sum up to the original image. This is not possible in NLP since there is no meaning in adding an integer to a word (e.g., “dog” + 1).
- *Human Perception.* From a human point of view, perturbations in CV are difficult to perceive, since the modifications are at a pixel level. Vice-versa, on text, small changes are easily detectable by both human beings and machines (e.g., spell checkers).
- *Semantic.* From the semantic point-of-view, the addition of a perturbation into an image rarely changes its meaning. In NLP, the modification/addition/removal of a character/word may lead to a completely different meaning of the sentence (e.g., “I hate you”, “I ate you”).

As a consequence, state-of-the-art attacks on NLP are either CV-algorithms adapted to face NLP challenges, or novel solutions designed from scratch.

In this work, we are mainly interested in the *evasion attack*. As previously introduced, our goal is to define a perturbation that influences the target model while preserving the semantic and readability of the sentence. A small amount of perturbation can guarantee a correct human perception; for example, in [230], authors show the human resistance to leet speech, e.g., “R34D1NG WORD5 W1TH NUMB3R5”. The choice of the measurement is not trivial as in CV, where spatial distance metrics between the original sample x and the malicious x' are used. As [325], we can measure the perturbation in different ways, such as norm-based distances for dense representations, or edit-based measurement, which identify the number of changes required by making x' equal to x .

7.2 Zero-Width Attack

In this section, we present the Zero-Width attack (ZeW). We first introduce in Section 7.2.1 the motivations and the intuition that drives our investigation. In Section 7.2.2 we describe how ZeW can affect different NLP pipelines. We conclude with Section 7.2.3 by describing a countermeasure to our proposed attack.

7.2.1 Motivations

Three main motivations guide our investigation.

1. *UNICODE representation.* Most NLP tools allow the use of UNICODE characters. This is essential, especially for the analysis of web text. For example, on Social Networks, non-ASCII characters are often used (e.g., emoji).
2. *Readability Preservation.* The attack strategy should apply fewer modifications as possible to maintain the sentence readability.
3. *Indexing stage vulnerabilities.* To the best of our knowledge, most of the attack strategies aim to leverage ML-models' weaknesses, while little attention has been put to the security of other stages of the text ML pipeline, such as the indexing stage (see Section 7.1.1).

We asked ourselves if it exists a technique that allows us to relax the constraint of the number of modifications to malicious sentences, allowing us to focus only on the disruption of target models' performance. We found the answer in the *steganography* discipline, which is the "art of hiding secret messages into plain sources" [23]. In the UNICODE representation, there are characters whose width is zero, i.e., when printed, they are invisible, and human beings cannot perceive them. Some examples of these characters are *zero-width space* (U+200B) and *zero-width joiner* (U+200C). These allow us to insert an arbitrary number of "invisible" characters in a given sentence. Thanks to this particular property, we can forget to consider the problem of readability preservation since sentences semantic is intact. The presence of zero-width characters allows an attacker to affect the decision of the indexing stage (see Section 7.1.1). We identify in total 24 malicious characters³.

In cybersecurity, we can find the usage of zero-width characters in different ways. For example, in [63] the authors use zero-width characters in the communication protocol of a botnet, ELISA; here, the botmaster secretly communicates through public posts with the zombies over social networks such as Facebook. In late 2018, the security team AVANAN discovered a phishing method against Office 365, bypassing Microsoft's security mechanisms [212]; in this attack, hackers used zero-width characters in the middle of malicious URLs, evading Microsoft's detection mechanisms. While in security zero-width characters are a known threat, to the best of our knowledge, we are the first to explore their effect in the adversarial machine learning context.

7.2.2 Theoretical Perspective

Zero-width characters give us the power to break the intra-relationship between the characters of a given sentence. Let's represent from now on zero-width characters with the symbol "\$". We can recall the example reported in at the beginning of this

³<https://github.com/pajola/ZeW/blob/main/ZeW.py>

chapter, i.e., “I hate this album”; the malicious version “I h\$a\$t\$e this album” appears identical to the original sentence from a human point of view, while different from a machine perspective. Figure 7.1 presents a real example of zero-width characters. We can notice that the malicious sentence appears legitimate.

In Section 7.1.1 we described possible numerical representations of a given sentence (*indexing* stage). We now explain how ZeW can affect those representations.

- *Word-based representations.* In word-based representations, a sentence can be seen as a temporal vector $s = t_0 + t_1 + \dots + t_n$, where t_i is the token (i.e., word, punctuation symbol) at the time i , and n is the length of the tokenized sentence (see Section 7.1.1, *preprocessing* stage). Here, it is unlikely that words containing “\$” are present in the vocabulary V . Two possible scenarios can occur.
 - Unrecognized words are mapped to special tokens (e.g., placeholders, “UNK”). It is likely that unpoisoned words and “UNK” have different meanings and effects to target models, since they appear with a different representation. For example, the sentence “I h\$a\$t\$e this album” is represented as “[I, UNK, this, album]”.
 - Unrecognized words are discarded from the analysis, with a consequence of loose of expressiveness of the malicious sentence. For example, the sentence “I h\$a\$t\$e this album” is represented as “[I, this, album]” (the word “h\$a\$t\$e” is discarded). In this case, the target model analyzes only the remaining sentence. Potentially, by adding one zero-width character per token, the resulting sentence will be empty.
- *Character-based representations.* In char-based representations, a sentence can be seen as a temporal vector $s = t_0 + t_1 + \dots + t_n$, where t_i is the character at position i , and n is the total number of characters that compose the sentence. As in the previous case, two possible scenarios can occur.
 - Unrecognized characters are mapped to the special tokens (e.g., placeholders, “UNK”), resulting in an addition of noise in the vectorial representation. For example, the word “h\$a\$t\$e” is represented as “[h, UNK, a, UNK, t, UNK, e]”.
 - Unrecognized characters are discarded from the analysis. In this case, the poisoned sentence coincides with the original sentence. The attack has no effect in this scenario. For example, the word “h\$a\$t\$e” is correctly represented as “[h, a, t, e]”.

In general, ZeW leads to an increase in noise or reduction of information in the sentence representation. The attack can be seen as an *injection* attack, where malicious characters are injected into target sentences. Potentially, an attacker can insert an arbitrary number of “\$” on malicious sentences, without any constraint. This gave us the capability of not considering the perturbation measurement described in Section 7.1.2. To the best of our knowledge, injection strategies using ZeW characters can be further optimized with target ML-models, resulting in the following adversarial attacks:

- *Evasion.* ZeW characters can be optimally inserted in target sentences to affect ML models’ decisions.
- *Poisoning.* If the adversary has access to the training data, the addition of malicious samples could lead to a noisy dataset, decreasing the overall performance.

- *Trojan*. If the adversary has access to the training data, he/she can inject a rare sequence of zero-width characters in a small portion of the dataset and let the model overfit over them. At test time, the trojan is triggered by samples containing that specific sequence.

The definition of such adversarial attacks is out of the scope of the paper.

7.2.3 Countermeasure

Overall, ZeW is an injection attack that influences the indexing stage (see Section 7.1.1), with consequences in the following steps (i.e., machine learning algorithms). ZeW leverages peculiar properties of UNICODE representation, which contains non-printable characters. In the security field, injection attacks are a standard and well-known problem [279]. A typical example is the SQL injection, where the definition of malicious input can damage the target database structure and destroy its contents. Injections can be severe, especially when users are allowed to insert arbitrary input used for critical operations. Similarly, MLaaS offer users to interact with ML-models through APIs. It is thus essential to have mechanisms that control any input feeding the models, placed at the *preprocessing stage*; these are also called sanitization or input validation mechanisms. Regarding ZeW, a simple filter that rejects malicious sentences containing non-printable characters is enough. Similarly, the sanitizer can just discard the malicious characters.

7.3 Case Study: Hate Speech Manipulation

In this section we introduce the designated case study (Section 7.3.1), followed by the injector algorithm description (Section 7.3.2).

7.3.1 Overview

We test and evaluate ZeW on text Machine-Learning-as-a-Service provided by popular companies such as Amazon, Google, IBM, and Microsoft. These services vary from sentiment analyzer to language translators. Our idea is to test some of the most popular ML-based text services to understand how many applications can be affected by the attack.

As a case study, we analyze the *hate speech manipulation*, a topic that raised the interest of a broad area of researchers in the last years [306, 259]. Our goal is to understand how zero-width characters affect the outcomes of different MLaaS. We consider the attack successful if the injection of zero-width characters affects in some way the performance of a target model. We are also interested to understand the magnitude of the vulnerability. In our opinion, this is a likely scenario where a malicious user aims to offend a target victim without being detected, since it is known the problem of malicious interaction between users and Artificial Intelligence systems. A famous example is the Microsoft chatbot Tay, which becomes hateful after a poisoning attack of a group of users [309].

7.3.2 Manipulation algorithm

In this work, we aim to define a simple yet effective strategy using ZeW attack. Simple and non-optimal attacks have been shown to be effective in [322], where

cybercriminals evade sexually explicit content detection with simple image transformations (e.g., random noise addition).

In our attack, we assume that hateful sentences contain a negative part-of-speech, as shown in Figure 7.3 on the *Real* corpus. We thus want to understand how the performance of the tested MLaaS are affected when “deleting” such negative parts. To do so, we designed a simple injection strategy that, given a sentence, identifies negative words (i.e., words with negative polarity scores) and injects on them zero-width characters. In the experiment, we inject zero-width characters in two possible fashions.

1. *Mask1*. Only one random Zero-Width SPace character is injected in the middle of the target word (e.g., *hate* \rightarrow *ha\$te*).
2. *Mask2*. Multiple random Zero-Width SPace characters are injected, one between each character (e.g., *hate* \rightarrow *\$h\$a\$t\$e\$*).

The idea behind these two strategies is to measure the impact of ZeW with different levels of injection. Algorithm 2 shows the overall attack strategy. To identify negative words we use *VaderSentiment*, a free sentiment analyzer tool available for Python [133]. The code of the injector is available on GitHub⁴.

Algorithm 2: HS-Manipulation.

```

input : An original sentence  $s$  and the type of injection mask  $m$ 
output: A poisoned sentence  $s_{pois}$ 
tokens = Tokenizer( $s$ )
 $N_{tok} = length(t)$ 
 $i = 0$   $s_{pois} = []$  while  $i < N_{tok}$  do
     $t = tokens[i]$ 
     $t_{stem} = Stem(t)$ 
     $t_{sent} = Sentiment(t_{stem})$ 
    if  $t_{sent}$  is negative then
         $t_{pois} = Injector(t, m)$ 
    end
     $s_{pois}.add(t_{pois})$ 
     $i = i + 1$ 
end
 $s_{pois} = Join(s_{pois})$ 

```

7.4 Results on Controlled Environments

In this section, we evaluate the impact of the ZeW injection strategy presented in Section 7.3 over different machine learning models and indexing techniques. In Section 7.4.1 we first present the experimental settings, followed by result discussions in Section 7.4.2.

7.4.1 Experimental Settings

Algorithm 2 aims to reduce the negative part-of-speech of a given sentence. We thus decide to understand the impact of ZeW injection strategy over a binary classification task: the sentiment classification. The task consists of predicting whether

⁴<https://github.com/pajola/ZeW>

a sentence is positive or negative. For the experiments we use the *Sentiment140 dataset* [104]. The dataset contains positive and negative tweets (800K per class), for a total of 1.6M of labeled tweets. We then randomly split the corpus into a training (70%), validation (10%), and testing partition (20%). We evaluate two types of ML algorithms:

- *SGDClassifier*. This is a linear classifier. We use Scikit-Learn [40] implementation. The model is built on top of a *TfidfVectorizer*, i.e., an engine that converts raw documents into TF-IDF representations.
- *Recurrent Neural Network Classifier*. We deploy standard RNN-based classifiers using an embedder, followed by a two layers GRU and a final linear layer. The model is deployed using Pytorch [227].

Each model is trained over different variants of text representation i.e., character and word-based. The SGDClassifier implements two different combinations: character ngrams defined in the range [1,5], and word ngrams, defined in the range [1,3]. For example, the range [1,2] means that the vectorizer considers unigrams and bigrams. The RNN classifier is defined over character and word unigrams tokenizer; in addition, we further consider RNN classifiers that use and discard “unknown” tokens. The models use a common and standard preprocessing technique that removes hashtag, mentions, and URLs from tweets. Table 7.1 summarize models’ configuration.

We now briefly describe the hyperparameters selection and training strategy of the two models categories. The SGD classifier is implemented with a greed search strategy over the following *TfidfVectorizer*’s hyperparameters: max document frequency (0.5, 0.75, 1), max number of features (1000, 5000), and use IDF (*True, False*). We use the validation set to find the best configuration. RNN models implement default hyperparameters configurations: the embedding dimension is 100, and the GRU’s hidden size is 256. The vocabulary size is set to 25K tokens for word-based cases, while 100 for character based ones; these vocabulary thresholds allows the model to learn the representation of “unknown” tokens. The training process uses Adam optimizer and BCEWithLogitsLoss as loss function. The models are trained for a maximum of 100 epochs. Note that we use a stopper mechanism that interrupts the training if a model does not improve its validation performance for 5 epochs.

7.4.2 Results and Considerations

In this section, we evaluate the performance of the six models presented in Section 7.4.2. The first evaluation is conducted with the *accuracy score* (ACC), i.e., the percentage of correct predictions. Table 7.1 summarize the results. As expected, DNN-based models tend to outperform simple linear models; this gap can be linked with the limited vocabulary size adopted in the *TfidfVectorizer* due to memory limitations. We also highlight that the usage of unknown tokens does not boost-up models’ performance.

The effect of ZeW is measured with the *attack success percentage* (ASP), i.e., the percentage of sentences classified as positive. Note that such a percentage also contains those samples that are misclassified in normal conditions. The evaluation uses three corpora: a set of original tweets called “*real*”, and two malicious counterparts (one per mask) named “*mask1*” and “*mask2*”, respectively. The set “*real*” corresponds to the negative test sentences (160K); we then discard those sentences that cannot be modified by Algorithm 2, resulting in a final set with 75K tweets.

Table 7.1 presents ZeW success percentage. We can notice that the ASR is always under 40%. This result can be explained with:

- a limitation of Algorithm 7.3.2, where the injection strategy modifies negative tokens. However, a sentence’s polarity might be the effect of a sequence of tokens rather than the sum of single instances polarity.
- a limitation of ZeW, where the injection is limited to a strict set of operations (i.e., the insertion of a set of characters).

Nevertheless, we can find some insights from such results:

1. ZeW can affect the performance of different models that use different tokenization strategies. The combination of ZeW with state-of-the-art attacks targeting ML-models can result in dangerous effects.
2. In general, character-based models are more resilient to ZeW. In particular, we highlight that when unknown tokens are discarded, ZeW attack fails.
3. In general, models that consider unknown tokens are more vulnerable. An attacker can thus leverage this factor.

TABLE 7.1: Overview of models’ performance. The accuracy score (ACC) measure the quality of the model on the three splits. The attack success percentage (ASP) measures the misclassification percentage of a given classifier; in bold the results of models resistant to ZeW.

| ML Model | Tokenization | UNK | Train (ACC) | Valid (ACC) | Test (ACC) | Real (ASP) | Mask1 (ASP) | Mask2 (ASP) |
|---------------|--------------|-----|-------------|-------------|------------|------------|-------------|-------------|
| SGDClassifier | Char | No | 77.15 | 77.00 | 77.19 | 12.06 | 22.15 | 29.63 |
| SGDClassifier | Word | No | 73.04 | 73.00 | 73.15 | 14.94 | 20.88 | 27.12 |
| RNN | Char | No | 81.68 | 81.52 | 81.46 | 5.27 | 3.72 | 3.72 |
| RNN | Char | Yes | 82.60 | 82.41 | 82.39 | 7.57 | 12.53 | 21.34 |
| RNN | Word | No | 84.79 | 84.20 | 84.28 | 6.93 | 37.75 | 37.19 |
| RNN | Word | Yes | 84.93 | 84.38 | 84.41 | 6.25 | 37.29 | 36.62 |

7.5 Results on MLaaS

In this section, we show how ZeW affects the performance of different MLaaS of the leading IT companies: Amazon, Google, IBM, and Microsoft. The considered companies provide similar services, and, where possible, the results are grouped-by. We identified the following macro-areas.

- *Hate Speech Detection* (Section 7.5.2). Tools that identify toxicity/hate speech in comments.
- *Insights Extractors* (Section 7.5.3). Tools that extract insightful information from the text (e.g., tones, personalities).
- *Sentiment Analyzers* (Section 7.5.4). Tools that measure sentence polarization.
- *Translators* (Section 7.5.5). Tools that translate sentences from a source language to a target one.

In this work, we do not compare the performance of ZeW with state-of-the-art since their focus is to exploit ML algorithms vulnerabilities, while we aim at the disruption of the indexing stage. Since our attack model is free of all of the restrictions in the number of modifications, an attacker can combine ZeW with attacks targeting ML algorithms.

7.5.1 Dataset & Evaluation on VaderSentiment

For the experiments, we use the hateful sentences available in [73], a well-known dataset of the task. This dataset contains three distinct classes: “hateful”, “offensive but not hateful”, and “neither” (nor hateful neither offensive). The dataset includes 1430 hateful sentences. We call now on the set of hateful sentences *Real*. Sentences that do not contain any negative word (detected) are discarded from *Real*. We then applied the injection algorithm with the two possible masks, generating two sets called, respectively, *Mask1* and *Mask2*. The final corpora contain 1094 samples each. All of the analyses and tests in different MLaaS use these corpora.

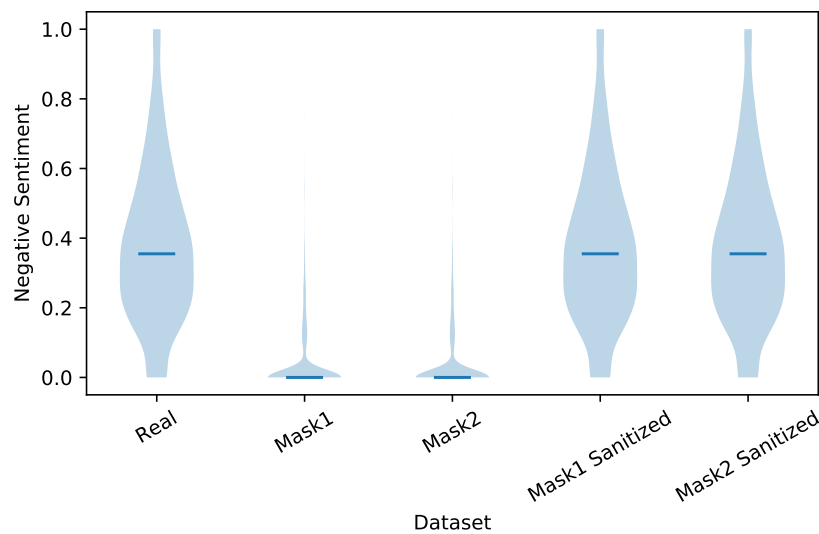


FIGURE 7.3: Negative sentiment densities of different corpus measured by VaderSentiment, where +1.0 is extremely negative and 0.0 is absence of negativity. The violin plot shows the distributions of the corpora’s negative scores; the blue line represents the median value of the distribution. The service is vulnerable if the distributions under attack are not equal to the distribution of *Real*.

We first analyze the impact of ZeW on VaderSentiment. As shown in Figure 7.3, both injection strategies (*Mask1* and *Mask2*) entirely cancel the perceived negativity. The median values of negativity scores are 0.35 (*Real*), and 0.0 for both *Mask1* and *Mask2*. ZeW is effective in both modalities against VaderSentiment. The injection of only one character per negative word is enough to disrupt this service. This might be a relevant problem since this tool is widely used in the scientific community.

In this section, we further show the effectiveness of our defense strategies proposed in Section 7.2.3, where the sanitization technique discards the malicious character from any given sentence. Figure 7.3 shows that the sanitized sentences have the same distribution of the original and unpoisoned corpus. Given the simplicity and the effectiveness of the proposed countermeasure, we decide to do not report similar results in the rest of this section.

7.5.2 Hate Speech Detection

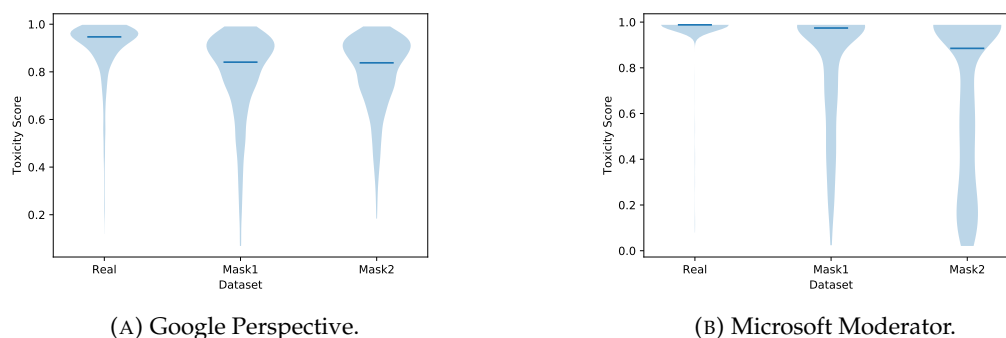
We start our analysis with hate-speech detection, the set of tools closer to our case study. These tools aim is to identify and detect the toxicity of sentences. The goal of an adversary is to write hateful sentences without being detected. This scenario is

likely on Social Networks (e.g., Facebook) that uses account suspension or ban when users post inappropriate content. We analyzed the following services.

- Google Perspective⁵. Perspective is part of the Conversation AI project, which aim is to improve the quality of online conversations with the supervision of ML. The tool identifies several aspects of online conversations that might be inappropriate, such as toxicity, profanity, and flirtation. In this experiment, we focus on toxicity manipulation, defined as disrespectful comments.
- Microsoft Content Moderator⁶. This ensemble of ML-tools aim to identify potentially offensive content in different type of media, such as text, images, and videos. Regarding the text-domain, the model identifies three categories of malicious content: sexually explicit content, sexually suggestive, and offensive. In this analysis, we focus on the latter. The tool offers the “autocorrect” option, which corrects grammatical mistakes before analyzing the contents. In our experiment, this parameter is set to TRUE.

Figure 7.4 shows the effect of ZeW on the toxicity detectors. Both services are highly resistant to the attack. On Google Perspective the median confidence level of the detector is 0.95 on (*Real*), 0.84 on *Mask1*, and 0.83 on (*Mask2*). Similarly, on Microsoft Moderator the median is 0.99 on (*Real*), 0.97 on *Mask1*, and 0.86 on (*Mask2*).

The impact of the attack is not strong, and the model seems resistant. On the other hand, in Google Perspective the insertion of only one zero-width character per negative word appears sufficient to damage the model’s confidence level. Similarly, the Microsoft tool can be affected by *Mask2*. We also need to highlight that the purpose of these tools is to detect high toxicity levels rather than detect negativity on sentences; thus, the algorithm described in Figure 2 might not be effective. The combination of ZeW with other state-of-the-adversarial techniques could seriously damage this service. We can state that both models are vulnerable to this attack.



(A) Google Perspective.

(B) Microsoft Moderator.

FIGURE 7.4: Toxicity score densities of different corpora measured by Google Perspective (left), and Microsoft Moderator (right), where +1.0 is high confidence of being classified as toxic. The violin plot shows the distributions of the corpora’s toxicity scores; the blue line represents the median value of the distribution. A service is vulnerable if the distributions under attack are not equal to the distribution of *Real*.

⁵<https://www.perspectiveapi.com>

⁶<https://azure.microsoft.com/en-us/services/cognitive-services/content-moderator>

7.5.3 Insights Extractors

Online Social Networks (OSNs) such as Facebook and Twitter are places where billions of users share their experiences, ideas, feelings, and opinions. These platforms are perfect for analyzing social behaviors and interactions. Several studies are conducted, from sentiment analysis and opinion mining [222, 251], to the prediction of when a security vulnerability will be exploited [50]. IBM offers two services that are helpful to analyze OSNs data. A possible attacker's goal is to hide his/her own personality.

- IBM Watson Tone Analyzer⁷. The tool detects and extracts emotional and language tones in a written text.
- IBM Watson Personal Insight⁸. The tool predicts the personality of a target user. For example, this tool allows us to analyze the tweets-history of a target Twitter account.

IBM Watson Tone Analyzer returns a list of emotions (strings) detected in a given sentence. Here, a possible adversary's goal is to hide/manipulate emotions from his text. To understand the efficacy of ZeW, we measured the similarity between the sets of emotions of the unpoisoned sentences and their poisoned counterparts. In particular, given a sentence x , its adversarial counterpart x' , and a Tone Extractor function f , we obtain the sets $A = f(x)$ and $B = f(x')$. The similarity between A and B is given by the *Jaccard Similarity*, defined as follows [216].

$$J(A, B) = \frac{\dim(A \cap B)}{\dim(A \cup B)} \quad (7.1)$$

where \dim returns the number of items in the set. The performance is measured by comparing the Jaccard similarities of *Real* vs. *Mask1* and *Real* vs. *Mask2*. Ideally, the API is resistant if the Jaccard Similarity is equal to +1.0 (two identical sets). In Figure 7.5, we can notice a different trend. The median values are 0.5 and 0.33 for *Mask1* and *Mask2*, respectively. A good portion of sentences (40%) are not affected; a possible explanation is that the negative words of those sentences are not essential to extract the emotion. Note that from this analysis we discard those sentences without any "tone" detected by the tool (322 sentences discarded).

On IBM Watson Personal Insight, the adversary's goal is to hide/manipulate his personality. In our test, we extract the personalities from the three corpora. In this experiment, the analysis is at a corpus-level rather than a sentence-level, i.e., we obtain one personality for each corpus. In Figure 7.6, we can notice that *Real* and *Mask1* differ in terms of "Openness" and "Conscientiousness", while *Mask2* seems to push all of the dimensions close to zero. In conclusion, both services of IBM are severely vulnerable to ZeW.

7.5.4 Sentiment Analyzers

Sentiment analysis is one of the most popular topics in NLP [224, 180, 3, 191] and can be used for several purposes, such as understand the opinions of restaurants, movies, or products. This importance is reflected by the fact that all companies

⁷<https://www.ibm.com/cloud/watson-tone-analyzer>

⁸<https://www.ibm.com/watson/services/personality-insights>

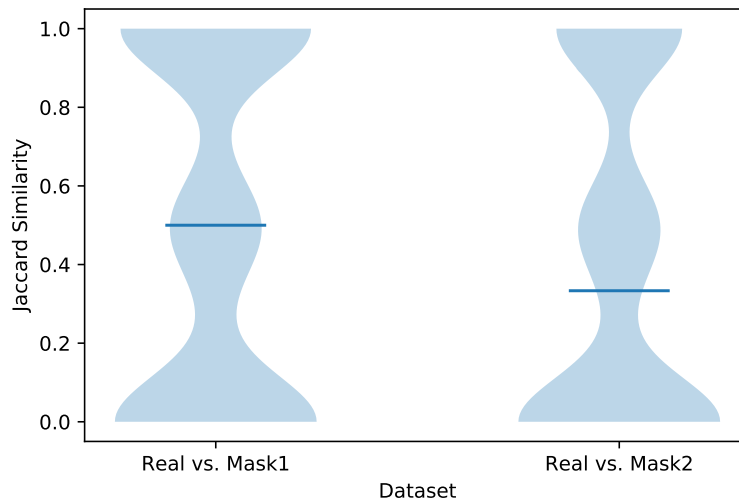


FIGURE 7.5: The similarity distributions between *Real vs. Mask1* and *Real vs. Mask2* of Watson Tone Analyzer, where +1,0 is an exact match between two sets. The violin plot shows the distributions of the corpora's Jaccard similarities; the blue line represents the median value of the distribution. The service is vulnerable if the distributions under attack are not close to one.

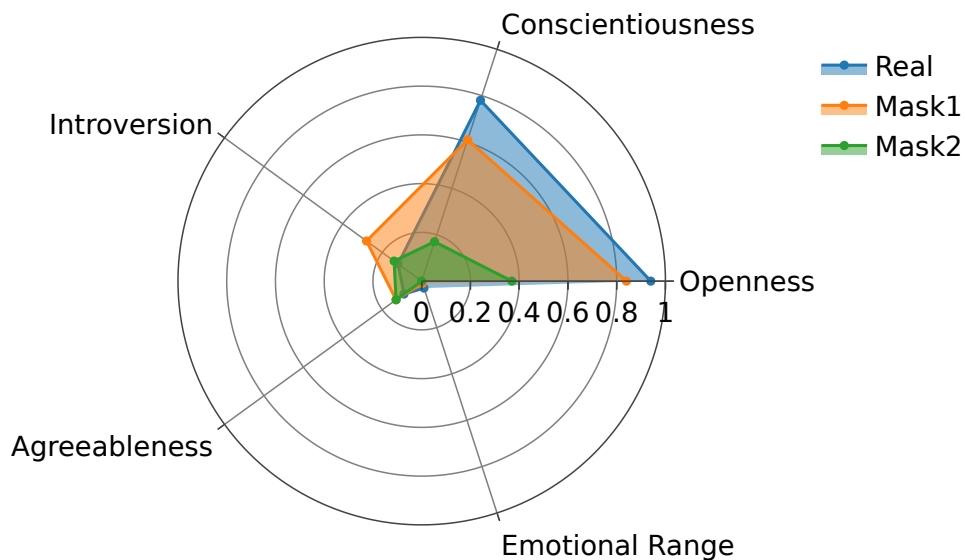


FIGURE 7.6: Watson Personal Insight detects three distinct personalities (*Real*, *Mask1*, and *Mask2*). The service is vulnerable if at least one of the five dimensions changes.

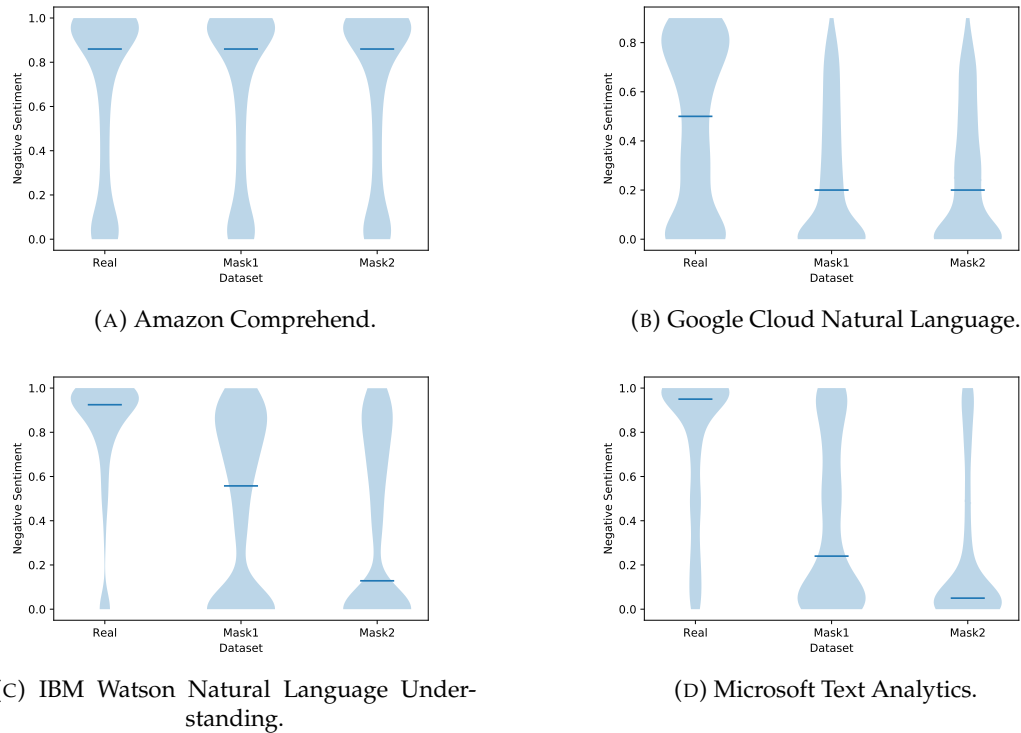


FIGURE 7.7: Effect of Zero-Width Space Attack on different sentiment extractor services. The violin plot shows the distributions of the corpora’s negative scores; the blue line represents the median value of the distribution. A service is vulnerable if the distributions under attack are not equal to the distribution of *Real*.

implement this service: Amazon Comprehend⁹, Google Cloud Natural Language¹⁰, IBM Watson Natural Language Understanding¹¹, and Microsoft Text Analytics¹².

In the hate speech scenario, as shown in Figure 7.3, the sentences are likely to be perceived as negative. A possible adversary’s goal is to minimize the detected negativity; this attack can be seen as a *transferable attack* [225] since our malicious sentences are first tested on a sentiment analyzer (i.e., VaderSentiment).

Figure 7.7 shows the effectiveness of ZeW on 3 services out of 4. In particular, Amazon Comprehend is resistant to both modalities of injection, where the median value is constant (0.86). Google Cloud Natural Language shows a similar vulnerability pattern for both masks, with an equal median value that moves from 0.5 to 0.2. In this service, the addition of one character per negative word is sufficient to disrupt it. We conclude with the services provided by IBM and Microsoft, where we see a common decreasing pattern of the median values, which move from 0.92 / 0.95 on *Real*, to 0.56 / 0.24 for *Mask1*, and to 0.13 / 0.05 for *Mask2*. We can state that three out of four services are severely vulnerable to ZeW, while only one show resistance.

⁹<https://aws.amazon.com/comprehend>.

¹⁰<https://cloud.google.com/natural-language>.

¹¹<https://www.ibm.com/cloud/watson-natural-language-understanding>.

¹²<https://azure.microsoft.com/en-us/services/cognitive-services/text-analytics>.

7.5.5 Translators

We conclude the results section with another well-known NLP task: the language translation. All four companies implement this service: Amazon Translate¹³, Google Translation¹⁴, IBM Watson Language Translator¹⁵, and Microsoft Translator¹⁶.

In the hate speech scenario, we can imagine that the adversary writes a hateful message in an unknown language for the victim. The victim uses translators to understand the meaning of the message. For example, human moderators could use automatic translators to understand if comments written in foreign languages are hateful. Another example is browsers like Chrome that automatically translates web content.

An example of this scenario is shown in Figure 7.1, where the malicious sentence “I wanna kill you” is translated as “I love you” by Google Translate¹⁷. Note that since the target model is unknown, we do not have any control over the target output. We highlight here that the aim of the attacker is to degrade the general performance of the target model rather than control the translation process. ZeW is evaluated on the translation task English-Italian. To understand the impact, we measure the similarity between the translations given by the unpoisoned sentence and its malicious counterpart. The difference is measured with the Bilingual Evaluation Understudy Score (BLEU score), with its 4-gram cumulative implementation. Formally, given a sentence x , its malicious counterpart x' , and a translation function f , the similarity is defined as

$$\text{similarity} = \text{BLEU4}(f(x), f(x')). \quad (7.2)$$

Ideally, a service is not affected if the translations of the original sentence and its malicious version are the same, resulting in BLEU score equal to +1.0 (perfect match). In Figure 7.8 we can see that all of the services are vulnerable to the attack. Amazon seems resistant to *Mask1*, with a median value equal to 1.0, while vulnerable to *Mask2*, with the median equal to 0.83. Similarly, IBM is resistant to *Mask1* and vulnerable to *Mask2*: the median value is 1.0 for *Mask1*, and 0.58 for *Mask2*. Google and Microsoft show vulnerabilities in both injection strategies, where the median values move from 0.63 / 0.47 in *Mask1*, to 0.40 / 0.34 in *Mask2*.

All of the models show more difficulties in handling *Mask2*. These tools show different vulnerability patterns compared to the sentiment analysis tasks. The possible explanation is the nature of translators: Seq2Seq models (i.e., autoencoders). Seq2Seq models likely use different placeholders to deal with OOV tokens, as introduced in Section 7.1.1. We can state that all of the services are vulnerable to ZeW: three strongly vulnerable, and only one weakly (Amazon).

7.5.6 Considerations

In this section, we analyzed how different MLaaS behave under the ZeW attack. We can notice different trends among types of services (e.g., sentiment analyzers) and the same companies (e.g., Microsoft). We now try to understand why these models behave differently.

ZeW seems to fail on *hate speech detectors*. This result suggests that both services use character-based tokenizers, which is a reasonable assumption since such

¹³<https://aws.amazon.com/translate>.

¹⁴<https://cloud.google.com/translate?hl=en>.

¹⁵<https://www.ibm.com/cloud/watson-language-translator>.

¹⁶<https://azure.microsoft.com/en-us/services/cognitive-services/translator>.

¹⁷<https://translate.google.com/>.

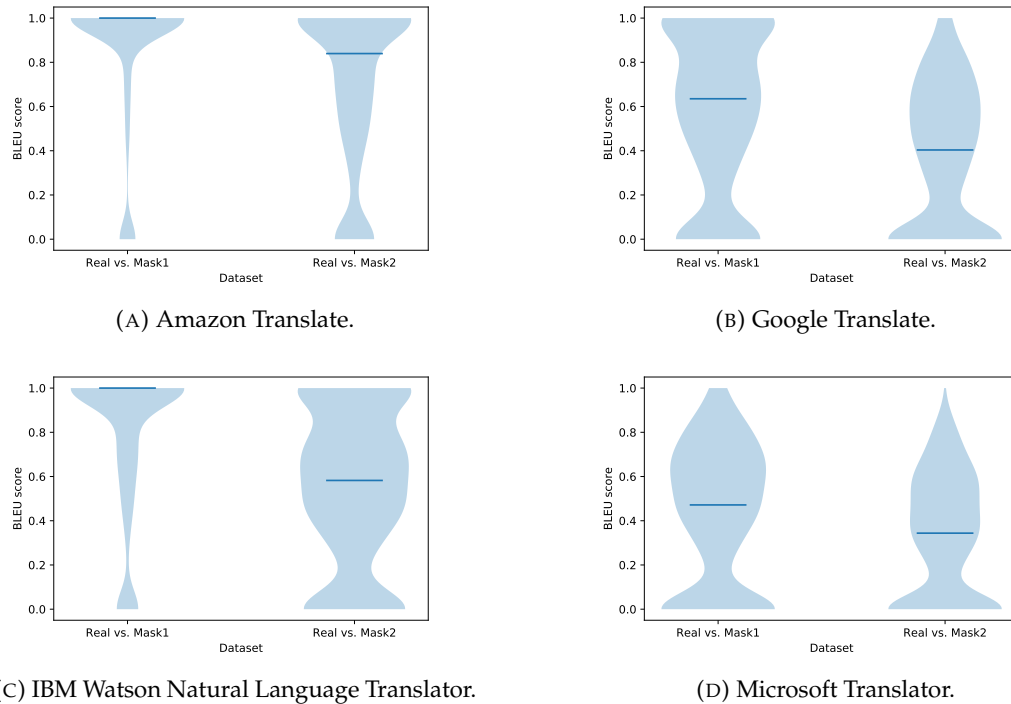


FIGURE 7.8: Effect of Zero-Width Space Attack on different translator services. The violin plot shows the distributions of the corpora’s BLEU scores; the blue line represents the median value of the distribution. A service is vulnerable if the distributions under attack are not close to one.

services should deal with noisy text (e.g., grammatical errors, misspelling) gathered from blogs, forums, and social networks. Moreover, such services are resistant to the injected noise (unknown tokens); a possible explanation is that these services deal with unrecognized words (e.g., discard). Amazon services show similar performance.

In general, IBM MLaaS are vulnerable to ZeW attack. Similar trends are shared among different services (e.g., Watson Personal Insight, sentiment extractor), where the attack is more effective when we inject more ZeW characters. These trends are similar to the RNN char-based with UNK performance, as shown in Table 7.1.

Finally, on translators, we find two patterns: i) resistant only to *mask1* (i.e., Amazon, IBM), and vulnerable to both injection levels (i.e., Google, Microsoft). Since *mask2* has a stronger impact, the four models might be character-based. However, it is unclear why there is such discrepancy, where two out of four models are resistant to *mask1* ZeW attack. More in-depth investigations should be conducted on neural machine translators architecture.

7.6 Related Work

In the state-of-the-art we can find several adversarial attacks targeting the machine learning algorithms of MLaaS. We now briefly summarize attacks on the MLaaS considered in Section 7.5.

Hate Speech Detectors Several scientific discussions use Google Perspective as a case study of their hate speech evasion techniques. For example, in [127, 115] the authors show the ability to manipulate Perspective by adding small mistakes to the sentences (e.g., typos, leet speech, word addition, word removal). In [38], the authors proposed evasion techniques based on acoustic and visual similarities, with an evasion power equal to 33% and 72.5%.

Sentiment Analyzers In [109], to manipulate sentiment tools, the authors applied techniques from computer visions, i.e., Fast Gradient Sign Attack [111]. In DeepFool [207], the authors manipulate the sentiment analysis of a CNN model. This algorithm uses Word Mover’s Distance (WMD) [166] to find suitable words whose embeddings allow to influence the target classifier. Similarly, in [8], the authors propose a word replacement algorithm based on semantic similarities. In [169], authors describe TextBugger, a black-box framework that achieves high evasion success rate on different Machine-Learning-as-a-Services.

Machine Translators Cheng et al. propose AdvGen, a gradient-based method for attacking Neural Machine Translation (NMT) models [57]. In [56], the authors propose two techniques to evade Seq2Seq models (e.g., translators) using ad-hoc loss functions: *non-overlapping attack* and *keyword attack*. For the first, the goal is to generate completely novel adversarial sentences, while for the latter, the malicious translation contains target keywords. For the interested reader, we suggest finding more details on adversarial machine learning in Seq2Seq models in [56].

7.7 Limitations

In this section we briefly discuss the limitations of ZeW attack and the proposed countermeasure.

Attack The results presented in Section 7.5 show how different commercial services can be affected by the proposed attack ZeW. However, the efficacy of ZeW is strictly related to services implementation choices. For example, as shown in Section 7.4, char-based models are more resilient compared to word-based ones. Moreover, when the model discards unrecognized characters, the attack is completely unsuccessful. Another major drawback is the limited control over malicious samples and, as a consequence, over the effect of the attack. If we consider language translators, an attacker can affect the translation, but he/she has no control over the output. For example, in the attack reported in Figure 7.1 we did not target that particular translation. Similarly, on the classification task, an attacker can only reduce the likelihood of a sentence being in a specific class (e.g., in this work we reduce sentences’ negativity) and not let the sample be classified as a target class.

Defense In Section 7.2.3, we present a simple yet effective countermeasure, consisting on the removal (sanitization) of zero-width characters from any given sentence. This choice is possible since normal English sentences should not contains such characters. Moreover, to understand if a ZeW attack is occurring, models owners can feed their applications with both original and sanitized sentences and look for results discrepancies. The proposed sanitization technique is however applicable only for ZeW attacks, resulting in a patch rather than a general solution. A popular countermeasure adopted in the state-of-the-art is the *adversarial training*, where,

for example, the defender augments the training data with examples of adversarial samples to make the model more robust [111]. Even though the adversarial training showed promising results, we believe that a strong and simple countermeasure consists of limiting applications' character vocabulary. We recall that our attack uses characters that are not normally present in the written language, and thus a simple input control can raise alerts whenever unlikely characters are identified. Finally, as reported in Table 7.1, character-based models present an intrinsic resiliency to ZeW attack; future commercial implementation should consider this aspect.

7.8 Summary

The migration of machine learning applications from research to commercial and industrial purposes increases the necessity of finding security mechanisms that guarantee the correct usage of them. In this work, we present a novel injection algorithm: Zero-Width attack (ZeW). This attack injects non-printable UNICODE characters on malicious sentences, with a potential disruption of the indexing stage of the ML application pipeline, while maintaining the full-readability of the text. This gives us the opportunity to do not consider the readability constraint, one of the major obstacles in the text adversarial machine learning field.

Our goal was twofold: i) understand how different pipelines respond to ZeW attack, and ii) whether commercial applications are vulnerable to ZeW attack. In Section 7.4 we showed that different implementations are vulnerable with different magnitude to the attack, while character based models show promising "security by design" patterns. We then demonstrate the ferocity of the attack on commercial solutions (Section 7.5): on 12 services developed by top IT companies such as Amazon, Google, IBM, and Microsoft, 11 show vulnerabilities. Among these 11, only 3 present a good resistance to the attack, while the remaining 8 are heavily affected. The simplicity of the attack allows it to spread to a broad population of malicious users and activities since no prior knowledge of machine learning theory is required.

Potentially, we can find several use-cases of our attack and not only hate-speech manipulation. For example, we can consider web data mining techniques that can be used for counter-terrorism [290] where, NLP technologies can help to identify malicious content. In this scenario, terrorists could use ZeW to obfuscate the contents of their web-pages, affecting the performance of the analyzer. Because of this, our simple but effective countermeasure based on an input-validation technique should be integrated into every real-life NLP tool.

The security of machine learning applications is strictly related to their input domain. Computer Vision has different challenges compared to Natural Language Processing, which has different challenges compared to the signal domain. Moreover, state-of-the-art mainly focus on the security of the machine learning models, by forgetting that a machine learning application is composed by several stages where the ML model is only one of these. In conclusion, we believe that novel malicious opportunities can be derived by exploiting vulnerabilities of different components of the ML pipeline, and one of these directions is the leverage of multiple representations of the text, such as the usage of ASCII and UNICODE.

Chapter 8

Captcha Attack: Turning Captchas Against Humanity

The field of adversarial machine learning (AML) studies the security of machine learning (ML) algorithms. Currently, AML literature examines theoretical aspects of the discipline with experiments conducted in artificial settings. There is, however, a lack of understanding of *how* adversaries approach ML systems in the wild [116]. Considering that real-life adversaries might adopt approaches that differ from what is described in state-of-the-art, this situation becomes even worse. It was discovered, for instance, that real-life adversaries adopted techniques less optimal than theoretical attacks, yet still effective, to promote pornographic content [322].

This work aims to study real-life adversarial attacks on Automatic Content Moderator (ACM). ACM are an essential element to guarantee a safe and secure environment for users of online platforms such as blogs, forums, and Online Social Network (OSN). Indeed, OSN such as Facebook and Instagram leverage human operator to identify and remove content that violate platform's policies, such as pornographic images or toxic posts (e.g., racist, hateful). Since users generate more content than humans can moderate, automated tools are needed to assist human operators. As reported by *TheVerge* [301], Facebook employs ML tools to monitor users' posts to spot potential inappropriate content that human operators will manually review. Such content is either removed or labeled as "sensitive", which means users have to explicitly accept to view it. Instagram recently adopted a similar system [137], stating that technology and humans co-operate to identify sensitive content (Figure 8.1).

ACM are an ideal case-study of real-life adversarial samples for different reasons. 1) OSN have a broad audience, with people coming from different background and cultures. Adversarial techniques thus might vary in their execution. 2) OSN popularity result in a massive share of daily content. For instance, Instagram and Facebook count about 350 thousand stories and 150 thousand photos posted by their users, respectively, every minute [272]. These "big numbers" increase our probability to find adversarial samples. 3) OSN adversaries likely adapt their sharing strategies to avoid automatic filters [322].

Contributions In this study, we focus on posts defined as images containing text (e.g., meme), a popular method of communication of OSN like Instagram. We present an analysis of 4600 popular Instagram posts from pages and hashtags containing potentially toxic text (e.g., hateful, sexually explicit). We discovered that 44% of them present obfuscations that might undermine ACM decisions. As these posts are reminiscent of captchas (i.e., not understandable by automated mechanisms), we

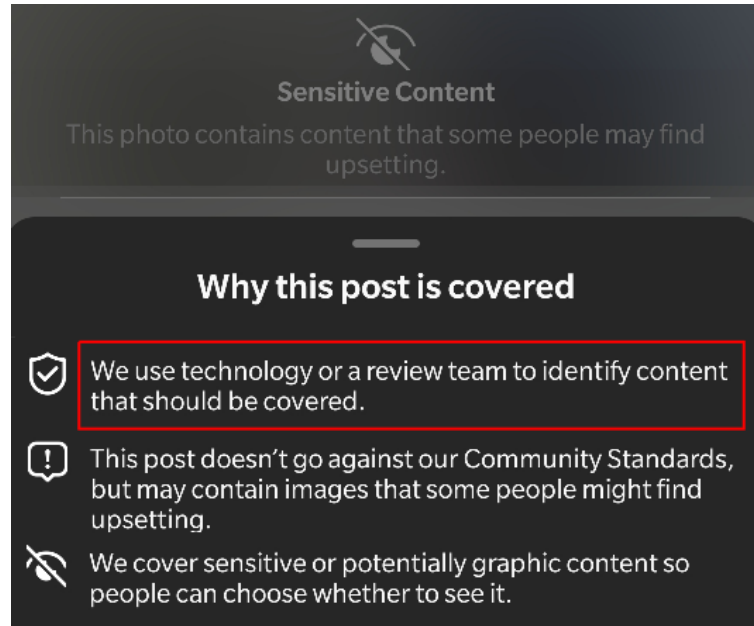


FIGURE 8.1: Instagram alert of sensitive content.

coin this threat as Captcha Attack (CAPA).¹ Figure 8.2 shows an example of a meme we produced using some of these obfuscations (e.g., typos, letters-shaped objects, hard background). Thus, we investigated this phenomenon and harvested such adversarial samples from the wild to define a taxonomy of obfuscation techniques. This contribution seeks to revisit the classical concept of captchas (*defenses* used in web platforms to distinguish humans from machines), and define a new type of captcha generated by humans to *evade* machine controls. Specifically, in our experiments, we focus on a broad CAPA sub-categories, consisting in samples adopting textual Captcha Challenges, namely CC-CAPA. We empirically demonstrate the ferocity of CC-CAPA by proving that current ACM cannot detect such samples, with an evasion rate equal to 100% (i.e., perfection).



FIGURE 8.2: Example of meme with different obfuscations (e.g., typos, letters-shaped objects, hard background).

Our investigation highlights that such failures are the result of a weak text extraction phase - conducted by Optical Character Recognition (OCR) - which is an essential step to handle images containing text. It is therefore crucial that OCR are

¹The name of our attack is a quote to Caparezza, an Italian singer famous for his lyrics rich with puns.

trained to deal with such obfuscations, but a large amount of data is necessary in order to accomplish this. Collecting data is not trivial, especially into the wild. We thus propose two CC-CAPA identification strategies: supervised and unsupervised. The former are ideal to identify CC-CAPA samples adopting known templates at training time. The latter is a solid methodology to spot unknown CC-CAPA templates, and more in general, new CAPA families. The effectiveness of our approaches is demonstrated through extensive experiments on three OSN: Pinterest, Twitter, and Yahoo-Flickr. We summarize our contribution as follows:

1. We define and propose a taxonomy of Captcha Attack (CAPA) – a set of obfuscation techniques adopted in OSN that undermines ACM.
2. We demonstrate CC-CAPA (a broad CAPA sub-category) extremely high evasion capability on real-world ACM.²
3. We propose two solid detection strategies to spot CC-CAPA and CAPA samples into the wild, conducting experiments on three OSN: Pinterest, Twitter, Yahoo-Flickr.

Organization The Chapter is organized as follows: Section 8.1 introduces background and related works. Section 8.2 presents CAPA taxonomy. Section 8.3 and Section 8.4 show the design of the attack and its results, respectively. Section 8.5 illustrates the detection strategies, and we conclude in Section 8.6.

8.1 Background & Related Works

This section presents theoretical concepts with related works required to understand the rest of the paper entirely. We discuss security of ML-based applications (Section 8.1.1), moderators in OSN (Section 8.1.2), and captchas (Section 8.1.3).

8.1.1 Security of Machine Learning Applications

ML applications like automatic content moderators need to deal with real-world challenges, offering at the same time high performance and attack resiliency. Therefore, when considering the application security, we need to consider all of the components of such pipelines, like preprocessing function, machine learning algorithms, and developing libraries (e.g., PyTorch, Scikit-learn). In general, an adversary's goal is to control and affect ML application decisions through the definition of *adversarial samples*.

Adversaries can affect ML applications by exploiting ML algorithms. We find different attacks such as the *evasion attack*, where the attacker defines malicious samples that fools a target classifier [25, 113], and the *poisoning attack*, where attackers affects model performances if they have access to the training data [22, 249]. On the opposite, adversaries can further exploit vulnerabilities derived by the ML application pipeline (e.g., libraries bugs, preprocessing functions) [314]. Such attacks are domain and application-related. For example, in the image domain, attackers can exploit image scaling techniques [313].

This work focuses on spreading adversarial sentences through images. Given captchas deceiving nature, we can categorize our attack as *cross-modal* against *Optical*

²We test ACM APIs to avoid the spread of inappropriate content.

Character Recognition (OCR) [325]. OCR are tools that extract text from images. Base-line adversarial attacks on OCR use different strategies like noise and watermark addition [51, 52]. These attacks are optimized to fool a target model. In contrast, our attack leverages captchas that are an antagonist of OCR by definition. Thus, the proposed attack CAPA is not optimized to fool ACM machine learning algorithms but rather to affect earlier stages, such as the text extraction from images using OCR.

8.1.2 Moderators in OSN

Online platforms use human moderators to monitor content shared in their virtual environment and block any malicious content before spreading. However, their efficiency is limited by the many users and interactions that a platform presents daily. To overcome this issue, companies started developing automatic tools. From [103], “the major platforms dream of software that can identify hate speech, porn, or threats more quickly and more fairly than human reviewers, before the offending content is ever seen.” For example, Facebook uses ML to flag potentially harmful content and remove automatically clear-cut cases, while the rest are processed by human operators [301]. A final contribution to the moderation process is made by OSN users: where human and automatic moderators fail, OSN users may report content that is offensive or harmful.

Human and automatic content moderators need to deal with multimodal content such as text, image, video, and audio. We can thus find several moderator tools based on the aim and source type. A popular and widely studied application is *hate speech detection*. Furthermore, online platforms are often visited not only by adults but by children as well. Image and video can contain contents that are not appropriate for such a young audience. Examples are *violent* and *sexually explicit* content detectors [281, 198]. While these tools mainly focus on textual contents with NLP-based solutions [259], only recently the attention moved on the multimodal representations (e.g., text inside images). For example, a new popular trend is the hateful meme detection [153, 108, 297], where the ACM combines images and textual information to address the task.

Given the variety of content, ACM need to deal with multiple sources and types of information. In this work, we focus on text and images, which can generate four types of content: (i) textual content like comments, (ii) image content like photos, (iii) images accompanied by text, like a photo with a caption, and (iv) images containing text like memes. Thus, an ideal ACM should contain DL modules that can work with text and images. The ACM workflow is straightforward in cases like i, ii, and iii. In contrast, for iv the workflow is more complex since the ACM should first extract through an OCR textual information; then, the DL components should process both textual and visual contents. The decision of content being toxic should thus consider both sources. Figure 8.3 shows such a pipeline. While different companies can adopt and develop different ACM, our described pipeline can still faithfully describe their workflows since we do not discuss how to implement specific operations [153, 108, 297].

8.1.3 CAPTCHA

A CAPTCHA (Completely Automated Public Turning Test to tell Computers and Humans Apart) is a test to distinguish between humans and computers (e.g., bots, automated users). First examples appear in 2000, designed by Von Ahn et al. [302], to check whether web requests were coming from humans, improving the security

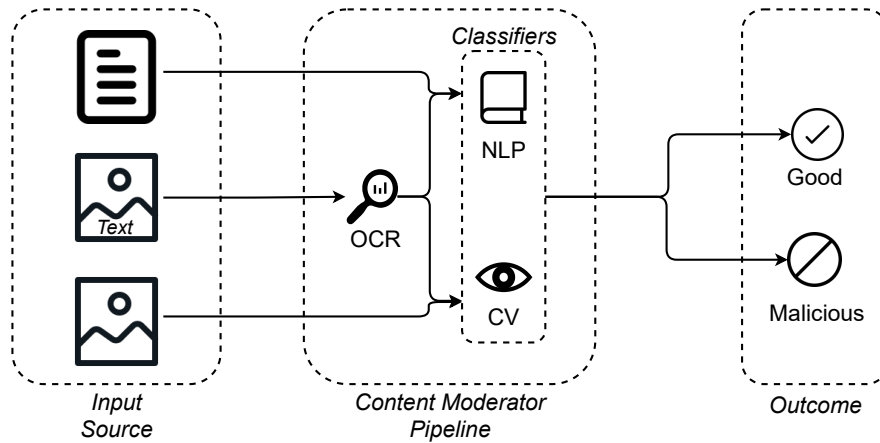


FIGURE 8.3: Overview of a content moderator in the text and image domains.

of websites, such as by preventing spam, protecting users' registration, and limiting email address scraping. The first generation of captchas was based on text, altered by rotations, distortions, or wavings, to be hardly readable by a machine (e.g., OCR) but simple for humans. With the advancements in AI technology, text-based captchas began to be solved, with a significant decline in 2014, when Google demonstrated that even the most complicated variants could be easily broken [112]. The security weaknesses related to text-based captchas led the research community to develop new techniques, e.g., based on images, audios, videos, or puzzles [268]. In general, their evolution follows the advancements of technology to break them [118]. Even if the text-based captchas security has been proved to be inefficient, they are still preferred by many users because of familiarity and sense of security and control [164].

The research community put much effort into solving (or breaking) text-based captchas (the type used in our attack). Their robustness has been shown to heavily rely on the difficulty of finding where the character is, i.e., *segmentation*, rather than what character it is, i.e., *recognition* [48]. The breaking methods evolved from algorithmic techniques [208, 316] to machine learning based approaches [41, 97, 112].

8.2 Captcha Attack: A Taxonomy

We now present Captcha Attack (CAPA) and a taxonomy of its variants in Section 8.2.1. We discuss the two macro-level of obfuscations: OCR-failures (Section 8.2.2) and classifier-failures (Section 8.2.3). We report statistics of the CAPA adoption on Instagram in Section 8.2.4.

8.2.1 Challenges from OSN's users: a Taxonomy

Finding potential adversarial samples in a platform such as Instagram is not trivial given its large amount of posts shared daily. Therefore, we limited our investigation to posts (images containing text) whose text can potentially be considered toxic. We focused only on posts in English or Italian, which we could fully understand since language can be a barrier to identifying elements such as typos, slang, or double meanings. We selected four popular hashtags, three well-known English meme pages, and three well-known Italian meme pages, all related to memes or adult content (potentially harmful). We limited our manual inspection to the latest 100 posts

for each page to analyze the most recent trends. Given that hashtags convey content from many pages and users, we focused on the latest 1000 English posts without incurring the risk of analyzing old content. A total of 4600 posts were manually analyzed as a result of this process.

We now need to define *what is a potential adversarial sample*. By considering ACM nature in the multimodal case, as previously discussed in Section 8.1.2, potential threats can be derived by:

1. **OCR-failures** – a wrong text extraction. We thus considered challenges inspired by captchas (Section 8.1.3), such as complex backgrounds or occluders.
2. **Classifiers-failures** – perturbations that can undermine NLP modules, such as typos and leet speech [115].

Using this criteria, we observed that 44% of the 4600 posts present at least one obfuscation.³ We thus decided to investigate the nature of such posts profoundly, and we organized the found obfuscation techniques, resulting in the CAPA taxonomy presented in Figure 8.4. The organization follow the security violation level, i.e., at OCR or NLP level.

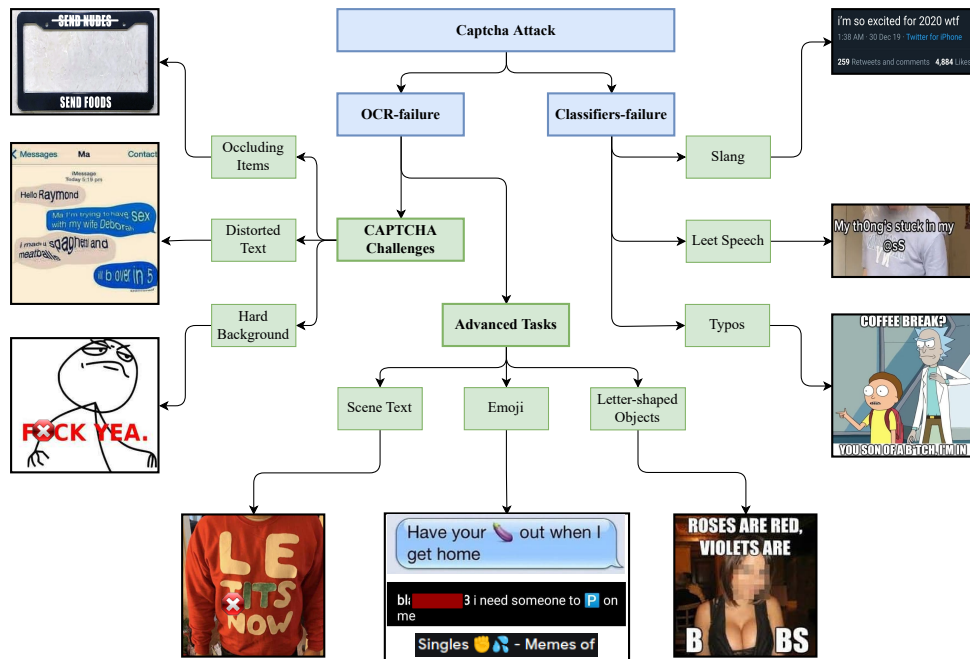


FIGURE 8.4: Obfuscation techniques we identified in online social networks. Blue boxes represents the ACM component that might fail. Green boxes represent different obfuscation techniques. We censored explicit or harmful contents with the red symbol.

8.2.2 OCR-failure

OCR-level obfuscations aim to disrupt or affect the text extraction phase from images. We identified two sub-family of techniques: *advanced task* for OCR and *CAPTCHA challenges*.

³We want to underline that, in this phase, we considered not only potential harmful posts, but any post published in such pages that might undermine the correct workflow of ACM.

Advanced Tasks for OCR

With *advanced tasks* we mean a set of applications that differ from the classic document extraction and pose more challenges for OCR. For example, *scene text* recognition is an area that gained popularity in the last few years [185]. This task consists of detecting and extracting text from real-life scenes (e.g., a road sign, T-shirt). Another exciting challenge is letter-shaped objects, i.e., images whose shape recall a specific alphabet letter. OCR might not recognize the correct character, resulting in an erroneous extraction. This task is not yet discussed in literature to the best of our knowledge. We conclude with the family of *emoji* obfuscations. In Figure 8.4, we show three typical examples of emoji obfuscations. On the top, the text contains an eggplant with a visual double-meaning (i.e., referring to a penis). In the middle, the P-emoji is used with a phonetic deception (i.e., P can be read as ‘pee’). On the bottom, two emoji are combined to represent a sexual action.

CAPTCHA Challenges

CAPTCHA challenges represent obfuscations usually adopted by textual captchas. Such transformations are *hard background*, *distorted text*, and *occluding items*. While we classified these obfuscations as a stand-alone, they are usually blended with other obfuscations we presented in the taxonomy.

8.2.3 Classifier-failures

ML-level obfuscations contains techniques that, while allowing a proper textual extraction, undermine the correct functioning of ML classifiers. These techniques are similar to those presented in [115]: slang, leet speech, and typos. The first category relates to post that contains slang terms (e.g., wtf → what the f*ck). The second class is the *leet speech*, where some characters are replaced with other visually similar ones, e.g., a * s → @sS. The last class relates to text with *typos* or grammatical mistakes, i.e., images containing misspelled words that, however, can be comprehended by human readers. In the example, we show a meme that contains a sentence with a swear word where the letter ‘i’ is replaced by ‘*’.

8.2.4 Statistics from the wild

Table 8.1 reports the statistics of CAPA usage in the wild, supporting our taxonomy. From the 4600 posts analyzed, we discovered that 44% present at least one obfuscation strategy. We can first notice that ‘hard background’ is present in most of the sources, reaching the 77% in one case. In general, this seems a trend of new posts, where the text is written on top of a complex background (e.g., real life scene). Moreover, we noticed that some techniques (i.e., emoji, leet speech, typos, and occluding items) were mainly used to cover sexually explicit content or swear-words.

Takeaway 1: Users are adopting adversarial techniques in OSN.

TABLE 8.1: Percentage of obfuscation techniques observed in different Instagram sources.

| Source | CAPTCHA Challenges | | | Advanced Tasks | | | Classifier-Failures | | |
|-------------------------|--------------------|------------|---------|----------------|-------|-----|---------------------|-------------|-------|
| | Occl. Items | Dist. Text | Hard BG | Scene Text | Emoji | LSO | Slang | Leet Speech | Typos |
| epicfunnypage | 3.0 | 10.0 | 14.0 | 9.0 | 5.0 | 0.0 | 26.0 | 1.0 | 4.0 |
| 6.memes.9 | 6.0 | 7.0 | 33.0 | 4.0 | 5.0 | 0.0 | 5.0 | 0.0 | 5.0 |
| 9Gag | 0.0 | 0.0 | 20.0 | 1.0 | 0.0 | 0.0 | 7.0 | 0.0 | 0.0 |
| partitodisagiato | 11.0 | 0.0 | 77.0 | 1.0 | 23.0 | 0.0 | 3.0 | 15.0 | 4.0 |
| pastorizianeverdiesreal | 0.0 | 0.0 | 29.0 | 2.0 | 1.0 | 0.0 | 1.0 | 0.0 | 0.0 |
| alpha_man_real | 1.0 | 1.0 | 47.0 | 1.0 | 0.0 | 0.0 | 1.0 | 0.0 | 3.0 |
| #naughtymemes | 5.3 | 4.1 | 23.8 | 5.2 | 12.1 | 0.2 | 9.7 | 3.0 | 5.4 |
| #sexualmemes | 3.2 | 6.1 | 23.2 | 4.7 | 7.3 | 0.0 | 15.2 | 0.4 | 5.5 |
| #nsfwmemes | 7.0 | 7.1 | 31.8 | 0.8 | 4.2 | 0.0 | 14.5 | 2.1 | 5.9 |
| #adultmemes | 1.3 | 3.2 | 18.4 | 5.9 | 4.7 | 0.4 | 10.0 | 4.3 | 5.3 |

8.3 Attack Execution

This section describes the ferocity of CAPA in real-life conditions. We start by motivating the attack (Section 8.3.1), followed by the generation procedure of adversarial samples and the resulting dataset in Section 8.3.2 and Section 8.3.3, respectively.

8.3.1 Motivation

Section 8.2 presented Captcha Attack (CAPA), i.e., examples of real-life obfuscations we spotted on social networks like Facebook and Instagram. Among these posts, we saw several extremely inappropriate (e.g., sexually explicit, hateful sentences) obfuscated with one or more techniques. Studying ACM behavior in the presence of such ‘adversarial’ samples would highlight ACM weaknesses. Behind these obfuscations, we always find the same rationale: people are trying to create content that can be easily understood by humans but is challenging for machines.

An ideal way to study how ACM would behave with these malicious samples would require collecting a vast number of them. However, we find three major challenges to collect such dataset: (1) these obfuscations seem novel and a direct consequence of the recent adoption of ACM in OSNs [301], resulting in a limited number of samples; (2) there are many variants or ways to produce an obfuscation, making the problem of limiting samples worse; (3) an automatic tool to detect such posts currently does not exist. In Section 8.5, we discuss in details a strategy to collect such dataset.

To address the previously listed issues and to effectively evaluate current real-world ACM robustness, we focus on the automatic generation of CAPA by leveraging classic textual captchas containing custom words. Custom textual captchas can be considered a broad sub-category of the more general class of CAPA presented in our taxonomy (Section 8.2, CAPTCHA challenges branch). The adoption of an automatic generation process present the following advantages. First, given a set of captcha styles, we can generate an arbitrary number of samples. Second, the generated samples represent a simplified version of real-life posts since they do not contain any visual aspect that might affect CV classifiers (e.g., racist visual components). Third, classic textual captchas have been widely investigated in the literature, and thus the knowledge acquired so far might help counter CAPA in this and all of its forms. Therefore, from now on, through our experiments, we deeply explore the CAPTCHA Challenges branch of CAPA, which we call CC-CAPA.

8.3.2 CC-CAPA Generation Procedure

This section describes the process of generating a custom textual captchas, i.e., CC-CAPA. Given an harmful custom textual sample x , and an automatic content moderator M , we aim to identify a transformation function T such that:

$$\begin{aligned} M(x) &= c_i, \\ M(T(x)) &= c_j, \end{aligned} \tag{8.1}$$

where c_i is the offensive class, and c_j the non offensive one. The function T should satisfy the following properties.

1. *Easy to deploy.* This would open to a broad target of possible adversaries, not only people highly skilled in computer science. CC-CAPA is easy to execute, as shown by the already deployed attacks we presented in Section 8.2.
2. *Target model agnostic.* The transformation should be independent of the target system, i.e., the process T is not mathematically optimized to fool a specific ACM M , but rather any ACM. This would make the attack stronger and effective to different unknown ACM. As we are going to discuss in the rest of this section, CC-CAPA does not require any information about the target system.
3. *Effective.* The attack should be successful with high confidence. This is desirable since online platforms follow strict policies for inappropriate content sharing, e.g., suspension or account ban. From a theoretical perspective, the usage of captchas should guarantee a high evasion rate. We demonstrate CC-CAPA effectiveness in Section 8.4.

The first CC-CAPA transformations T is the insertion of text in images. This domain-change transformation T_1 represents the first deceptive layer. While analyses on the text and image contents follow standard predictions, the case of text contained in images might represent a gray area since it involves additional operations such as text extraction and the cooperation between NLP and CV machine learning algorithms. If an online platform does not explicitly develop an ACM handling such cases, there is a high chance that malicious content $T_1(x)$ can evade detection mechanisms. We explore this scenario in Section 8.4.2.

If we consider proper implementations of automatic content moderators (see Figure 8.3), setting $T = T_1$ might not be sufficient to guarantee complete attack effectiveness. Thus, the addition of typical manipulation and distortion of classic textual captchas produces images with similar properties of the ones presented in Figure 8.4. For example, we noticed most of posts (e.g., memes, Instagram reels) present an hard background. A customizable textual captcha can be seen as a function composition:

$$T = T_n(\dots(T_2(T_1(x)))) \tag{8.2}$$

where T_1 represents the domain transfer function, while the set $[T_2, \dots, T_n]$ is the combination of image transformations to generate the captcha, and x is the given sentence. As reported in [320], popular transformations can be applied at the background (e.g., solid, complex, noisy), character (e.g., font, size, color, rotation, distortion), and word level (e.g., character overlapping, occluding lines, waving, noise). The notation presented in Equation 8.2 can also describe generic CAPA images. Figure 8.5 shows an overview of the attack execution. The generation process we just described is well-known to the state-of-the-art. While this process does not constitute a part of the novelty of this work, in contrast, the usage of captchas from

defense solutions to attack vector in OSN is, to the best of our knowledge, not explored.

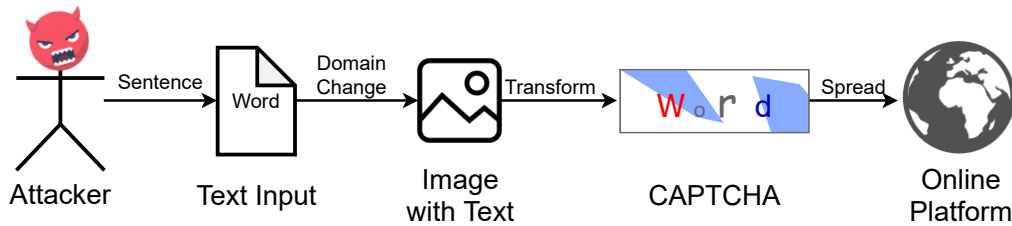


FIGURE 8.5: Overview of CC-CAPA execution pipeline.

CC-CAPA can exploit the following target ACM weaknesses:

1. *Unimplemented detection case.* The implementation of cross-domains ACM is not trivial and is not widely explored in literature. ACM not implementing such a scenario will miss images with harmful plain text.
2. *Text extraction phase.* If ACM deploys the monitoring of multimodal contents, a pipeline key phase is the text extraction. OCR usually handle this operation. OCR extraction from textual captchas might result in noisy inputs that feed NLP models and thus affecting their predictions.

8.3.3 CC-CAPA Dataset

As introduced in Section 8.1.2, a popular and essential ACM role is the identification of hateful messages on online platforms. Thus, an example of a possible attacker's goal is to let hateful messages being undetected by ACM. We build our dataset with potential hateful textual captchas. We retrieve a list of frequent English words associated with hateful sentences from *Hatebase.org* [120], for a total of 1383 samples. From this list, we maintain only those samples that, as stand-alone, should be banned from online platforms. For the purpose, we used *Microsoft Content Moderator*⁴ and *Google Perspective* as our ground truth⁵. These APIs identify the presence of different toxicity aspects. We first applied Microsoft moderation, obtaining 502 toxic words. We refined the list further using Google Perspective, producing a final list of 197 toxic words.

In this work, we are interested in understanding if ACM are vulnerable to textual captchas, particularly if different styles of textual captchas affect such target systems in different ways. We thus generate four variants of custom textual captchas. Each style differs in the type and number of transformations applied to the textual captcha. The four classes show different readability difficulties; the more transformations we apply, the more complex the image readability. We now describe the four adopted styles.

1. *Clean.* These are normal white images containing text. No further transformations are applied. Font: FreeMono.
2. *Claptcha.* Python captcha generator available on GitHub.⁶ Complex transformations are applied to the text. Font: FreeMono.

⁴<https://azure.microsoft.com/en-us/services/cognitive-services/content-moderator>

⁵For both APIs, we use 0.5 as threshold.

⁶<https://github.com/kuszaj/claptcha>

3. *Multicolor*. Python captcha generator available on GitHub.⁷ We modified the library to use arbitrary text of arbitrary length. Complex transformations are applied to the text. Font: Free family fonts.
4. *Homemade*. Our captcha generator, it aims to be more readable than *Claptcha* and *Multicolor*. Simple transformations are applied to the text. Font: FreeMono.

Figure 8.6 shows examples of attacks, one per class.

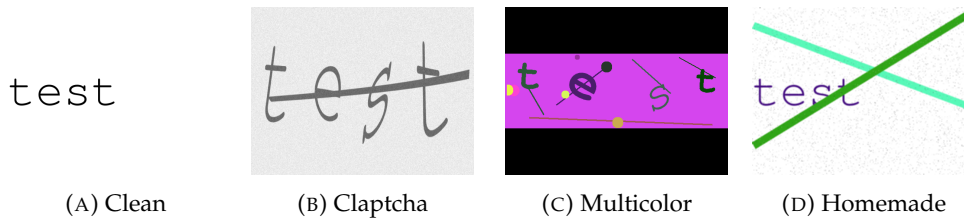


FIGURE 8.6: Captchas' styles used in the experiments.

The four sample classes are composed by different transformations. For example, *Clean* only uses only domain transfer, while *Claptcha* and *Multicolor* a high number of transformations. More details about the transformations are available in Appendix 8.7.1.

We produce *Clean* samples to verify if current deployed ACM deal with textual captchas, while we produced *Claptcha*, *Multicolor*, and *Homemade* to verify if attackers can affect ACM OCR. About the domain transfer transformation, this operation is easy to implement, from graphic software (e.g., Paint, Photoshop) to standard programming libraries (e.g., matplotlib). We further remark that there exist several online tools aiming to generate customisable textual captchas. The aftermath is that even attackers with low computer skills can produce customisable undetectable textual captchas. This statement is true if we consider that users are already producing CC-CAPA (and more in general, CAPA) samples, as discussed in Section 8.2.

Starting from the 502 toxic words identified by Microsoft, and 197 by the addition of Google, we produced two toxic captchas datasets: $|\mathbb{D}_{tox}^m| = 2008$ and $|\mathbb{D}_{tox}^{m+g}| = 788$, where m stands for Microsoft, and g stands for Google. These final datasets are validated through a user study which proves that the generated samples can be read very easily by human beings, thus supporting the idea OSN users would notice the malicious content even if written with captchas. The user study methodology and results are available in Appendix 8.7.1. We do not make the dataset publicly available since it might be used for attacks in the real world. However, we make it available upon requests for researchers to facilitate future investigations in this field.

8.4 Attack Results

This section presents the results of our attack in real-life scenarios. We first discuss the attacking scenarios we consider (Section 8.4.1), followed by a presentation of the results of our attack against already deployed ACM (Section 8.4.2) and against ACM following the schema shown in Figure 8.3 (Section 8.4.3).

⁷<https://github.com/J-Rios/multicolorcaptcha>

8.4.1 Overview

Based on the discussions of ACM deployment done in Section 8.1.2, we aim to verify the following:

1. Do current ACM consider cross-domain samples (e.g., text inside images)? We answer this question in Section 8.4.2 by attacking image moderators with *Clean* samples. We recall that these samples do not contain any transformation and, thus, OCR should successfully extract their text.
2. Are ACM considering cross-domain content vulnerable to CC-CAPA? Section 8.4.3 answers this question by analyzing ACM responses on *Clapcha*, *Multicolor*, and *Homemade* samples.

Tests of the ACM of social networks (e.g., Facebook) are not possible because it would imply the spread of inappropriate and harmful content. Furthermore, we cannot test the attacks to current state-of-the-art solutions (e.g., hateful memes detection) because, to the best of our knowledge, they all require that the text is successfully extracted through OCR [153, 108, 297]. Moreover, the hateful images presented in our dataset \mathcal{D}_{tox}^{m+g} (see Section 8.3.3) contain only hateful text, while the rest of the background is not harmful. Thus, we opted to test already deployed ACM APIs provided by top IT companies. Note that these services are already adopted from real systems and websites, as can be seen in the APIs presentation pages.

8.4.2 Image Moderators

Are current image ACM considering text inside images? To answer such question, we analyze the scores of only *Clean* samples. An example of *Clean* image is shown in Figure 8.6a. We test the following ACM deployed by top IT companies: *Amazon Content Moderation*⁸, *Google Safe Search Detection*⁹, and *Microsoft Content Moderator*.¹⁰ Each ACM identifies different aspects of inappropriate images. More details are available in Appendix 8.7.2. For each analyzed ACM, we consider a post malicious if it is linked maliciously to at least one of the malicious classes. We measure the attack performance with the *attack success rate* (ASR), defined as the ratio of unsafe content undetected divided by the total number of tests.

We find that all of the services cannot detect offensive text in images, even without obfuscation. *Clean* images reached a success rate of 1 for Amazon and Microsoft, and 0.97 for Google. The 3% images labeled as inappropriate by Google were identified as *spoofed*. This finding suggests that analyzed CV-based ACM do not consider the case of images containing text. We highlight the gravity of such a finding: if an online platform adopts current ACM solutions, attackers could bypass their automatic monitoring systems by *just* putting plain text inside images. Thus, online platforms should manually design defense mechanisms that follow the schema shown in Figure 8.3. We believe that the ACM developers should address this issue since leaving uncovered our proposed scenario (text inside images) weakens their systems' reliability, and expose their users to real threats.

Takeaway 2: *Real world ACM are not considering text within images, which opens severe security threats.*

⁸<https://docs.aws.amazon.com/rekognition/latest/dg/moderation.html>

⁹cloud.google.com/vision/docs/detecting-safe-search

¹⁰<https://azure.microsoft.com/services/cognitive-services/content-moderator/>

8.4.3 Cross-domain Moderators

Section 8.4.2 shows that ACM currently do not consider text inside images for the moderation. The natural follow-up question is: “Would adding an OCR module to an ACM effectively ban CC-CAPA?” We thus implement an ACM following the concepts introduced in Section 8.1.2 and preventively assess its robustness. In particular, we defined a pipeline that, given an image, extracts the text using an OCR, and then a textual ACM processes it. In this experiment, we vary the OCR technology while using Microsoft Content Moderator to spot potential harmful extracted sentences. In this stage, since we know a priori that Microsoft considers words $\in \mathbb{D}_{tox}^{m+g}$ as toxic, a misclassification after the text extraction can be due only to an OCR failure. We analyze OCR provided by Amazon¹¹, Google¹², Microsoft¹³, and the popular free python library Tesseract.¹⁴

We evaluate textual captchas with two metrics: the *attack success rate* (ASR) as defined in Section 8.4.2, and the average *normalized Levenshtein distance* (NLD):

$$NLD(x, x') = \frac{\mathcal{L}(x, x')}{\max(|x|, |x'|)}, \quad (8.3)$$

where x represents the true string in the image, x' the OCR output, \mathcal{L} the Levenshtein distance, and $|x|$ the number of characters in x . The Levenshtein distance measures the number of single-characters edits (e.g., addition, modification, deletion) required to make $x = x'$; it is defined between 0, when $x = x'$, and the maximum length between the two strings when they completely differ. The NLD measure defined in Equation 8.3 is thus defined in $[0, 1]$. With the ASR we aim to understand the evasion power of our proposed attack, while with the NLD we aim to understand the number of mistakes that OCR does.

Figure 8.7 shows the attack performance among the four services. We can first notice that the ASR rate on *Clean* images is very low, meaning that OCR correctly extract the input text. We recall that *Clean* samples do not have any visual transformation (e.g., rotation, complex background), and thus we expect that OCR work properly in such a case. This result suggests that ACM following the schema proposed in Section 8.1.2 are resistant to those attacks that only apply the domain-transfer technique T_1 . Moreover, such a schema present a valid solution easily adoptable by commercial ACM. Indeed, the results on *Clean* images are much higher compared to the one presented in Section 8.4.2.

On the opposite, the ASR is close to 1.0 for both *Claptcha* and *Multicolor* variants, meaning that offensive textual captchas successfully evaded the ACM in all samples. Our captcha implementation *Homemade* has an average ASR of 0.8, probably due to the less number of transformation applied compared to *Claptcha* and *Multicolor*. Similar trends can be found with the NLD measure. We report the complete results in Appendix 8.7.2

The results presented in this section suggest that ACM using the schema proposed in Figure 8.3 are vulnerable to textual captchas with few transformations (e.g., *Homemade* class). Moreover, the more transformations, the higher the attack success rate, reaching the perfect evasion rate for *Claptcha* and *Multicolor*.

¹¹<https://aws.amazon.com/it/textract>

¹²<https://cloud.google.com/vision/docs/ocr>

¹³<https://docs.microsoft.com/en-us/rest/api/cognitiveservices/contentmoderator/imagemoderation/ocrfileinput>

¹⁴<https://pypi.org/project/pytesseract>

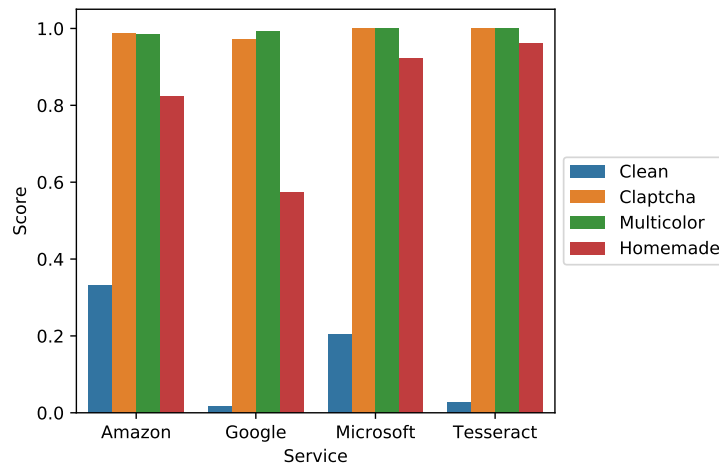


FIGURE 8.7: Cross-domain evaluation. We report the Attack Success Rate (ASR) (the higher, the better).

Takeaway 3: *Industrial OCR struggle against even the simplest obfuscation techniques.*

8.5 CC-CAPA Detection Strategies

This section presents CC-CAPA detection strategies. Section 8.5.1 describe possible defense directions. Section 8.5.2 and Section 8.5.3 present, respectively, supervised and unsupervised approaches to tackle the problem. In Section 8.5.4, we discuss general CAPA prevention. Last, we compare our defense strategies to state-of-the-art in Section 8.5.5.

8.5.1 Overview

In the previous section, we demonstrated how textual captchas can successfully evade ACM monitoring. Since the generation process of customizable textual captchas is quite naïve, this attack could be massively adopted by many users aiming to spread online messages without being censored. Indeed, social network users are already adopting CC-CAPA (and CAPA in general, see Section 8.2). For simplicity and to demonstrate the proposed attack capabilities, we tested the hate speech evasion task only. Even so, the attack surface is greater than evading hate speech alone, since it can encompass the entire spectrum of text that an online platform could potentially ban (e.g., opinion mining) or analyze (i.e., censorship).

It is therefore necessary to discuss potential mitigation to CC-CAPA. We identify three possible directions:

- *Prevention.* Making ACM robust to CC-CAPA is the ideal solution, which involves the deployment of more robust OCR.
- *Detection.* Strategies that identify CC-CAPA samples might help OSN to ban such samples or collect data to train robust OCR.
- *User Reporting.* After CC-CAPA spread in the OSN, users can manually report the presence of toxic content. This data can thus be used to train robust OCR as well.

Although ideal, prevention seems currently unpractical. Indeed CV researchers are currently studying solutions to make OCR robust in complex scenarios [179]. Similarly, user reporting should only compensate failures of automatic defensive mechanisms. Indeed, the reported toxic samples has already been spread and (potentially) harmed users. Moreover, we cannot estimate *if* and *when* users will report a toxic content.

In this work, we thus focus on the detection scenario. As we previously introduced in Section 8.1.3, captchas are generally a defensive mechanism. So far, the research community has primarily focused on the definition of new captchas or captchas breakers from an attacker’s perspective. The aftermath is that adopting textual captchas as an attack vector creates an uncovered area of cyber security: the captchas identification. Indeed, captcha breakers start from the hypothesis to know a priori if an image is a captcha [320]. OSN can adopt CC-CAPA detectors in three fashions:

1. Detected samples can be directly blocked; this solution might be useful when a platform requires absolute control over its content. Conversely, it might not be ideal in more relaxed scenarios. Indeed, if the platform decides to ban all CC-CAPA samples indiscriminately, users’ (inadvertently) posting benign CC-CAPA samples would feel censored without a reason.
2. Detected samples can be posted, but human operators will revise their goodness.
3. Detected samples can be gathered to create a dataset aiming to build OCR robust to CC-CAPA. We discuss this scenario in Section 8.5.4.

8.5.2 Supervised Approach: Classification

Overview. A simple solution is to distinguish CC-CAPA samples from normal OSN posts. In Section 8.5.1 we motivated the need of a countermeasures to our proposed attack, and we identified a possible solution: the *textual captcha identification*. We can model such a task as a binary classification problem, where the two classes are *captcha* and *non-captcha*.

Dataset. We now describe the datasets we used to deploy our defense, keeping in mind the following reasons:

1. *The target are OSN.* We must remember that, generally, ACM are deployed on OSN (e.g., Facebook, Twitter, Flickr). It is thus fundamental that the *non-captcha* class captures representative data of the target OSN.
2. *Imbalanced dataset.* Intuitively, we might expect that the majority of the posts in an OSN are not CAPA samples. Thus, we expect the dataset to be imbalanced and that the *non-captcha* class contains the majority of the samples.

We built three datasets, starting from three distinct OSN for the *non-captcha* class: Pinterest, Twitter, Yahoo-Flickr. We selected these datasets because images are a substantial portion of their daily content. For the *captcha* class, we used the dataset the authors created in [320], made out of 11 different schemes, each with 700 samples, for a total of 7700 samples. We call this dataset C11. In Appendix 8.7.2, more information about the captcha schemes contained in C11. Table 8.2 summarizes the statistics of the four sources. We thus created the three datasets: Pinterest + C11, Twitter + C11, Yahoo-Flickr + C11. Each dataset’s version is split using 70%, 10%,

TABLE 8.2: Datasets’ statistics.

| <i>Origin</i> | <i>Class</i> | <i>#Samples [k]</i> |
|--------------------|--------------|---------------------|
| C11 [320] | captchas | 7.7 |
| Pinterest [107] | non-captchas | 70 |
| Twitter [296] | non-captchas | 470 |
| Yahoo-Flickr [289] | non-captchas | 137 |

and 20% for the training, validation, and testing partitions. Due to computational limitations, we used just a random subset of Yahoo-Flickr.

Models. In this work, we use two type of models: *naïve classifiers* and *fine-tuned classifiers*. The naïve classifier is defined as follows: Conv2D with kernel size = 5 and 6 output channels, followed by a second Conv2S with kernel size = 5 and out 16 output channels; the output is then flattened and forwarded to three linear layers (10K neurons, 1000, and 2 respectively). Each layer adopts the ReLU as the activation function; moreover, after both Conv2D we apply a MaxPool2D with kernel size = 2. For the fine-tuned models, we use three well known pre-trained models: Alexnet [163], Resnet18 [283], and VGG [267]. The experiments are conducted in Pytorch. The fine-tuning strategy follows the official Pytorch tutorial [136]. All models are trained using an SGD optimizer (learning rate = 0.001, momentum = 0.9), a cross-entropy loss, and an early stopping mechanism that stops the training if the validation loss is not optimized for five epochs. The models are trained for a maximum of 200 epochs. **Results.** We evaluate our models using three standard metrics: F1-score macro, precision, and recall. Table 8.3 summarizes the results. In general, all of the classifiers obtain strong classification results close to 100% F1-score in all the scenarios (i.e., Pinterest + C11, Twitter + C11, Yahoo-Flickr + C11). This result implies that companies can easily recognize captchas schemes known at training time with extremely good performances.

TABLE 8.3: Avg retrieval results of 11 captcha schemes in different OSNs.

| <i>Dataset</i> | <i>Pinterest + C11</i> | | | <i>Twitter + C11</i> | | | <i>Yahoo-Flickr + C11</i> | | |
|----------------|------------------------|--------------|-------------|----------------------|--------------|-------------|---------------------------|--------------|-------------|
| | <i>F1</i> | <i>Prec.</i> | <i>Rec.</i> | <i>F1</i> | <i>Prec.</i> | <i>Rec.</i> | <i>F1</i> | <i>Prec.</i> | <i>Rec.</i> |
| Naïve | 99.8 | 99.3 | 99.9 | 99.5 | 99.2 | 99.2 | 99.9 | 99.6 | 100 |
| Alexnet | 99.9 | 99.8 | 1 | 99.8 | 99.5 | 100 | 99.9 | 99.8 | 100 |
| Resnet18 | 100 | 100 | 100 | 99.9 | 99.7 | 100 | 99.9 | 99.7 | 100 |
| VGG | 99.9 | 99.9 | 100 | 99.9 | 99.6 | 100 | 99.9 | 99.8 | 100 |

We further investigated if our supervised technique could spot captcha schemes not seen in training phase (i.e., unknown). Indeed, the definition of new captcha schemes is relatively easy by just varying the number and type of transformations. Moreover, a specific type of transformation can be executed differently; for example, occluding symbols can vary (e.g., lines, segments). As experiment, we tested our models on \mathbb{D}_{tox}^m (see Section 8.3.3), over the three classes *Claptcha*, *Multicolor*, and *Homemade*, which were unknown at training time. Our algorithm could detect some *Claptcha* samples, but failed with *Multicolor* and *Homemade*. A possible explanation is that *Claptcha*’ style is quite similar to some captchas styles presented in our training partition. Thus, if a platform is interested in finding unknown templates, a more generalizable defense solution is needed, which we present in Section 8.5.3.

Takeaway 4: Platforms can use supervised techniques to spot samples belonging to a target template, with extremely high accuracy.

8.5.3 Unsupervised Approach: Outlier Detection

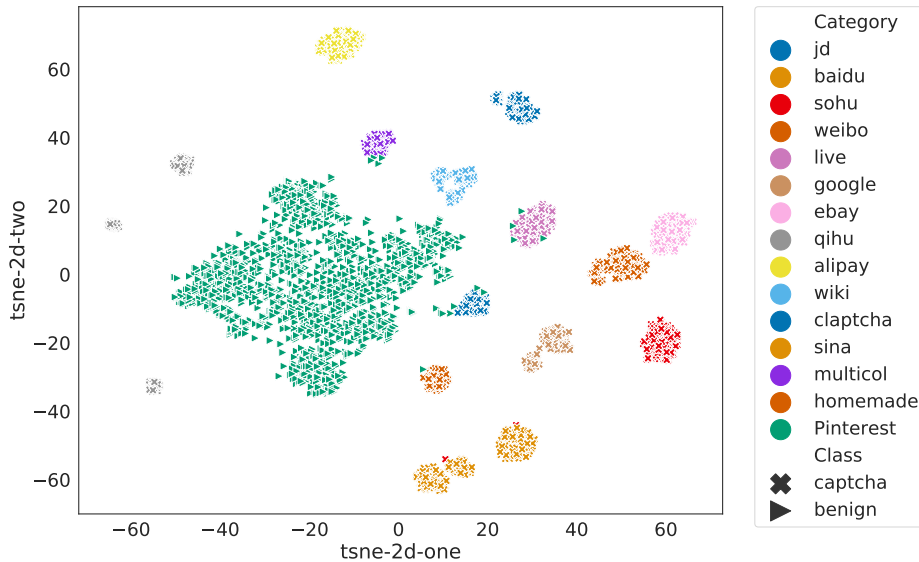
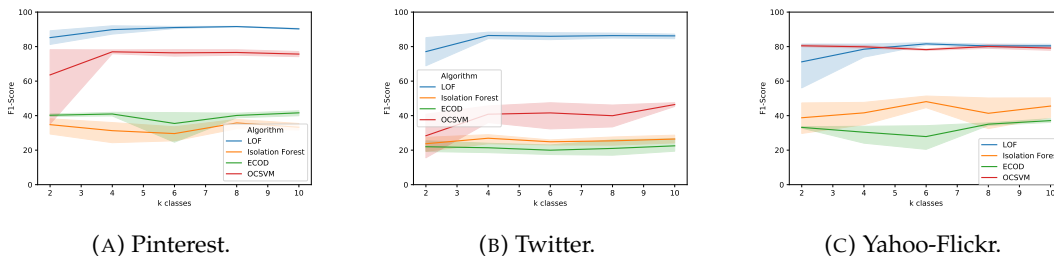


FIGURE 8.8: T-SNE 2D visualization of 2000 samples benign (Pinterest) and 2000 captchas (C11 and CAPA).



(A) Pinterest.

(B) Twitter.

(C) Yahoo-Flickr.

FIGURE 8.9: F1-score of different Outlier Detection at the varying of the OSN.

Overview. Supervised techniques guarantee high detection performance on known captcha schemes, while they poorly generalize on unseen styles. Therefore, for unknown styles, we adopt an orthogonal perspective toward our problem. We can assume that CC-CAPA is not widely exploited on the web platforms, and thus CC-CAPA samples look different from the majority of regular platforms' posts. Therefore, we adopt an outlier approach, where regular platforms posts are inliers and captcha outliers.

Dataset. We use the sources of the same datasets presented in Section 8.5.2 (Pinterest + C11, Twitter + C11, Yahoo-Flickr + C11) but a different training and validation strategy. In particular, the training set contains only samples belonging to the target OSN, while validation and test sets contain both benign and captcha samples. For each OSN, we first take a random subset of 50K samples, and then we split it into training (70%), validation (10%), and testing set (20%). In our investigation, we

are willing to understand *how many captcha styles* we should know to build a robust defense. Thus, we vary the number of known captcha styles in the validation set based on the 11 classes available in the C11 dataset. We experiment with different k known styles, $k \in \{2, 4, 6, 8, 10\}$. For each scenario, we repeat the experiment with 5 different styles combinations. The known captcha styles are then randomly split into validation and testing sets, with a 50% of proportion. The unknown captcha styles will belong exclusively to the testing set. Last, we add \mathbb{D}_{tox}^m to the test set (i.e., all the *Claptcha*, *Multicolor*, and *Homemade* captchas).

Models. All images are first converted into a 512-dimension embedding representation, using the pre-trained model ResNet-18 [283]. The first component of our defense is a dimensionality reduction module. We opted for the Principal Component Analysis (PCA), on which we vary the number of components: [2, 8, 64, 128]. We then tested the following algorithms: Isolation Forest (IF), Local Outlier Factor (LOF), ECOD [175], and One-Class SVM (OCSVM), using the implementation available in PyOD [330]. For each model, we tune a common hyper-parameter, i.e., the contamination level [0.1, 0.05, 0.01]. Moreover, IF are tuned on the number of estimators [16, 32, 64, 128], LOF on the number of neighbors [2, 4, 8, 16], OCSVM on the kernel type [*rbf*, *sigmoid*]. All the models are tune with a grid-search strategy.

Results. We first visually analyze our data, to better understand possible outcomes. Consider the combination of Pinterest, C11, and \mathbb{D}_{tox}^m datasets. We randomly sampled 2000 items each. From these samples, we first extracted the embedding, and obtained a two dimensional feature space with the combination of a PCA (from 512 to 50 features) and T-SNE (from 50 to 2 features). Figure 8.8 shows the distribution of 2000 Pinterest benign samples among different captcha samples. We can notice that captchas samples have distinct and unique pattern compared to Pinterest ones. However, each captcha styles defines an own and distinct cluster as well, explaining the poor generalization performance in classification tasks.

Our next step is to examine the results of three outlier detectors. Figure 8.9 shows the F1-score at testing time at the varying of the number of known captcha styles used in the validation set. LOF outperforms both Isolation Forest, ECOD, and OCSVM in the three OSN scenarios, reaching, on average, a performance of 80% F1-score. We can also notice that the amount of known styles has a limited impact, finding a performance stabilization starting from 4 styles. Furthermore, we identify consistent trends with both known and unknown captcha styles recognition. More details in Appendix 8.7.3. Going in more details, LOF labeled as outliers all the captcha classes (both C11 and our three classes in \mathbb{D}_{tox}^m), with a minimum of 60% accuracy for *Claptcha*, and up to 96% accuracy for *Homemade*. Thus, through this algorithm, we were able to identify all the captcha schemes through a generalizable solution.

In our experiments, our best algorithm (i.e., LOF) labeled as outliers about 5% of benign posts per dataset, that one could consider as false positives. However, since these posts are considered “unconventional” in the scope of OSN by our algorithm, there is an high chance that such posts could contain some other CAPA templates, or more in general, content that could have evaded ACM. Thus, we decided to perform a visual inspection on the three datasets. We manually inspected the 506 Twitter images, 530 Pinterest images, and 479 Yahoo-Flickr images labeled as outliers (i.e., false positives), which confirmed our intuition, i.e., CAPA is present in many variants:

- **Pinterest.** Only text without obfuscation: 22%; Text over hard background: 25%; Distorted text: 10%; Occluders: 3%; Natural Scene Text: 7%; Object shape: 1%.

- **Twitter.** Only text without obfuscation: 36%; Text over hard background: 36%; Distorted text: 14%; Occluders: 5%; Natural Scene Text: 11%; Emoji: 3%.
- **Yahoo-Flickr.** Only text without obfuscation: 1%; Text over hard background: 3%; Distorted text: 3%; Occluders: 1%; Natural Scene Text: 7%.

We also found 13 extremely dangerous images on Twitter (8 porn images, 3 gore images, one image inciting racism, and one image advertising drugs). Moreover, we found several posts adopting other CAPA templates. In Figure 8.10 we show some examples of false positives we identified through outlier detection techniques, which should require human moderation. In conclusion, through the presented unsupervised approach, we were able to spot all our captcha templates, and even additional CAPA templates (see taxonomy of Figure 8.4), such as all the CAPTCHA Challenges, Scene text, Emoji, and Slang obfuscations.

Takeaway 5: *Unsupervised algorithms offer a solid solutions to spot CC-CAPA. Among the outliers, additional CAPA templates emerged.*

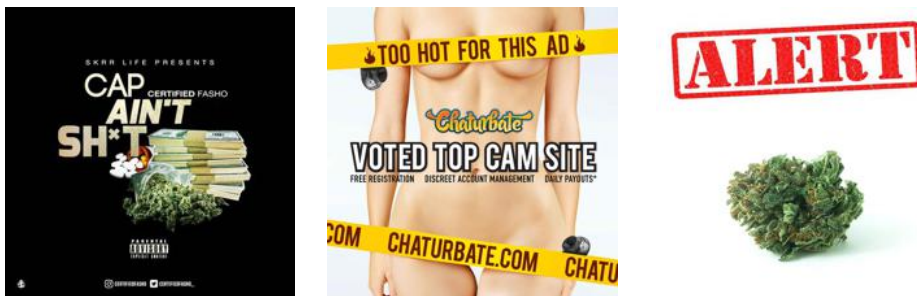


FIGURE 8.10: Example of false positive found among the outliers that should be moderated.

8.5.4 Toward Preventing CAPA

The presented methods can effectively detect CC-CAPA in the wild, offering a reliable defense strategy. The next step is to shift from *detection* to *prevention* techniques to defend from CC-CAPA, and more in general, from CAPA. The main reason we could not focus on prevention techniques, i.e., implementing and training robust OCR, is that a large dataset currently does not exist. Nevertheless, our supervised and unsupervised approaches can assist in reaching such a goal. We can thus shift the research question from *how to defend against CAPA* to *how to design a large enough dataset with CAPA samples*. Once we answer the latter question, researcher will need to focus on how to design effective OCR. The definition of a CAPA dataset can be divided into two stages:

1. *Identification*, aiming to identify CAPA families that can affect OCR.
2. *Retrieval*, aiming to collect a large number of samples belonging to a specific CAPA family.

Identification. The identification stage can be addressed by *user-reporting*, and *unsupervised techniques*. User reporting a toxic post implies a failure of the ACM, that can be derived by an OCR or toxic classifier failure. If OCR failure occurs, then the OSN identify a potential effective CAPA family. On the opposite, OSN

human operators can leverage *unsupervised techniques* (see Section 8.5.3) to identify anomalous posts shared in the platform, seeking for potential toxic posts that evaded ACM. Indeed, as shown in Figure 8.8, CAPA families differ from normal OSN posts.

Retrieval. Once human operators identify new family of CAPA samples, the goal is to collect a large number of similar samples that uses such obfuscations. The goal is to create a dataset with such a family containing toxic and benign samples. Here, they can use the supervised method we discuss in Section 8.5.2.

8.5.5 Comparison with State of the Art

Given the novelty of the attack in OSN, to the best of our knowledge, there are no other defensive detection methodologies to compare our work with. However, in the literature we found a similar topic to ours, i.e., detecting image spam conveyed in emails, sometimes resembling captchas [26]. In the survey of ten years ago that marked the problem of filtering image spam as “solved” [27], three main defense families were presented. The first family involves the usage of OCR extraction combined to text categorization, which is ineffective in our case, as discussed in Section 8.4.3. The second and third families are, respectively, *image classification* and *near-duplicate detection*: both focus on spotting spam images similar to templates known a priori. In particular, they first extract low-level features from a very specific template, and then use machine learning or statistical tests to find images similar to a query image. Through our experiments, we already demonstrated that if the template is “being a captcha”, both methods are ineffective. Indeed, captchas can have very different low-level features based on the adopted obfuscation techniques, and it is impossible to extract pre-determined features for the infinite number of possible obfuscations. Such approaches are better related to our scenario in which we know a captcha schema, and we find images belonging to it. In our experiments, we assessed a near-perfect detection in such situation. Moreover, our approach adopts Deep Learning (i.e., CNNs) to automatically extract low-level features (compared to manual extraction in prior works), making our methods more scalable and thus superior. For such reasons, we did not conduct further experiments based on prior works.

8.6 Summary and Future Works

Content moderators are essential in our society for the moderation of inappropriate content spread and shared on online platforms. Dangerous content (e.g., hateful words, nudity images) can potentially reach a broad audience, hurting or harming sensitive people. Online platforms started adopting automatic tools based on deep learning solutions to deal with the massive content volume.

As part of this work, we first present Captcha Attack (CAPA) and its taxonomy, which is based on observation of OSN obfuscated posts. We then experimentally demonstrate the ferocity of CC-CAPA, a broad sub-category of CAPA, showing that current ACM cannot moderate such samples. We demonstrated how easily an attacker could elude ACM detection by i) changing the domain from text to image and ii) applying captchas schemes. With the first, an attacker can evade those ACM not considering images containing text scenarios. With the latter, an attacker can affect NLP-based tools’ performance by exploiting OCR weaknesses. While CC-CAPA, and more in general CAPA, is easy to implement and does not require any

information about the target model, an ideal countermeasure is far from trivial. Toward this direction, we propose two solid detection approaches that can help to find CC-CAPA (and CAPA) samples in the wild.

Our work poses several challenges that might inspire future works. First of all, it is necessary to define a boundary between captchas and non-captchas. Second, for the various categories of the proposed taxonomy, a proper dataset should be collected, to eventually train detectors or sanitizers to help ACM. Last, it would be ideal to build a model that works against all the obfuscation variants described in taxonomy (e.g., emoji, leet speech).

8.7 Appendix

8.7.1 CAPA Dataset

Transformations

Table 8.4 summarizes the transformations applied in the four sample classes; we can notice that different styles adopt different transformations. For example, *Clean* only uses only domain transfer, while *Claptcha* and *Multicolor* a high number of transformations.

TABLE 8.4: List of transformations for textual captchas variants.

| <i>T#</i> | <i>Transformation</i> | <i>Clean</i> | <i>Claptcha</i> | <i>Multicolor</i> | <i>Homemade</i> |
|-----------|-------------------------|--------------|-----------------|-------------------|-----------------|
| T_1 | Domain transfer | ✓ | ✓ | ✓ | ✓ |
| T_3 | Rotation | | ✓ | ✓ | |
| T_4 | Distortion | | ✓ | | |
| T_5 | Waving | | ✓ | | |
| T_6 | Solid background | | | ✓ | |
| T_7 | Noisy background | | ✓ | | ✓ |
| T_8 | Different fonts & sizes | | ✓ | ✓ | |
| T_9 | Different colors | | | ✓ | ✓ |
| T_{10} | Occluding symbols | | ✓ | ✓ | ✓ |

Captchas Readability

Ideally, if CC-CAPA samples are posted on the web, they should be easy to read for humans, otherwise the whole attack would lose its purpose. Although posting unreadable content would surely evade any ACM, our samples need to have a good balance between low OCR-readability and high human-readability. While we evaluate efficacy of OCR in Section 8.4, we assess our captchas human-readability through an user study. We did not use any harmful word at this stage to not hurt anyone sensibility.

Methodology From a list of of English verbs¹⁵, we randomly selected 600 words, generating the corresponding custom textual captchas, 200 for each captcha class (*Claptcha*, *Multicolor*, *Homemade*). Then, we asked 50 participants (27 females, 23 males, age mean 28.6, std 6.1) to annotated them. Each candidate annotated 50 samples, providing the text they could read along with a difficulty score, from 1 (very

¹⁵github.com/aaronbassett/Pass-phrase

easy) to 5 (very difficult), to express how much the participant was sure about the answer, and how immediate the captcha was to solve. The confidence score is crucial to understand if people are likely to read captchas while scrolling social networks feed, or would ignore them because considered difficult. Each sample was processed by five participants. During the task, no time restrictions were given.

Metrics Participants are evaluated with two metrics: accuracy and *Character Error Rate* (CER). In particular, the accuracy evaluates the percentage of samples that were correctly annotated. CER, which is a popular OCR evaluation metric [200], measure the character distance between the annotation and the ground truth (the lower, the closer are the two words). The CER score is computed with Fastwer python library.¹⁶

Results As shown in Figure 8.11, we confirm the high readability of our samples. On average, humans obtained 94.53% and 1.31% of accuracy and CER, respectively. Overall, the task was trivial, with low difficulty scores reported (*Claptcha* = 1.3, *Multicolor* = 1.5, *Homemade* = 1.3). Moreover, we counted the number of samples that have been always successfully (or unsuccessfully) annotated by participants, producing and agreement score. Most samples are always correctly annotated (82.83%), while only 0.5% are always wrongly annotated. We thus expect a comparable high readability on CC-CAPA dataset as well.

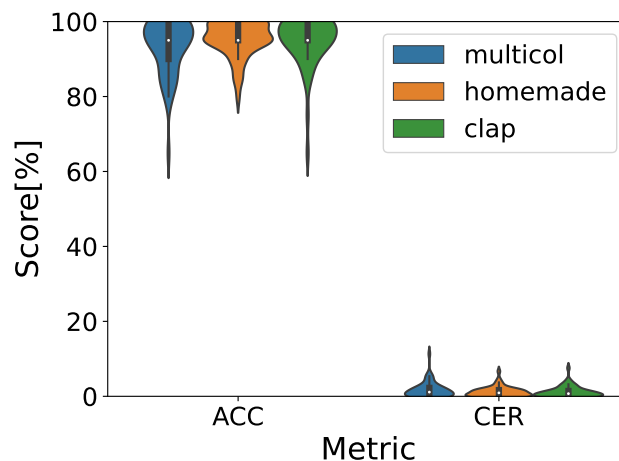


FIGURE 8.11: User-study performance distribution. We report accuracy (the higher, the better), and CER (the lower, the better).

8.7.2 Captcha Schemes











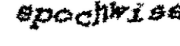
During our experiment, we used the dataset from [320] to represent the captcha class. Table 8.5 shows examples of them along with the applied transformations.

Attack

We test the following ACM deployed by top IT companies.

¹⁶github.com/kahne/fastwer

TABLE 8.5: Captcha schemes used in our experiment coming from [320].

| <i>Scheme</i> | <i>Example</i> | <i>Trasformations</i> |
|---------------|---|--|
| Alipay |  | Overlapping, rotation, distortion |
| Baidu |  | Occluding lines, overlapping, rotation, distortion, waving, varied font size & color |
| eBay |  | Overlapping, distortion, rotation, waving |
| Google |  | Overlapping, rotation, distortion, waving, varied font sizes & color |
| JD |  | Overlapping, rotation, distortion |
| Microsoft |  | Overlapping, solid background, rotation, waving, varied font styles & sizes |
| Qihu360 |  | Overlapping, rotation, distortion, varied font sizes |
| Sina |  | Overlapping, rotation, distortion, waving |
| Sohu |  | Overlapping, complex background, occluding lines, rotation, varied font size & color |
| Weibo |  | Overlapping, occluding lines, rotation, distortion |
| Wikipedia |  | Overlapping, rotation, distortion, waving |

- *Amazon Content Moderation*.¹⁷ The tool aims to classify inappropriate images among different classes, i.e., explicit nudity, suggestive, violent, visually disturbing, rude gestures, drugs, tobacco, alcohol, gambling, and hate symbols.
- *Google Safe Search Detection*.¹⁸ The tool returns the likelihood of content containing spoof, medical, violent, or racy content. The likelihood is defined with the following classes: unknown, very unlikely, unlikely, possible, likely, and very likely. We consider content malicious if it is classified as possible, likely, or very likely.
- *Microsoft Content Moderator*.¹⁹ The API identifies if the given image is appropriate for an adult audience (e.g., sexually explicit) or racist.

In Figure 8.12, the performance of CC-CAPA among different targets.

8.7.3 Defense

Table 8.3 summarizes the results of the classifiers when the captcha styles are known. Table 8.6 shows the generalization performance over the three \mathcal{D}_{tox}^m captcha classes: *Claptcha*, *Multicolor*, and *Homemade*.

¹⁷<https://docs.aws.amazon.com/rekognition/latest/dg/moderation.html>

¹⁸cloud.google.com/vision/docs/detecting-safe-search

¹⁹<https://azure.microsoft.com/services/cognitive-services/content-moderator/>

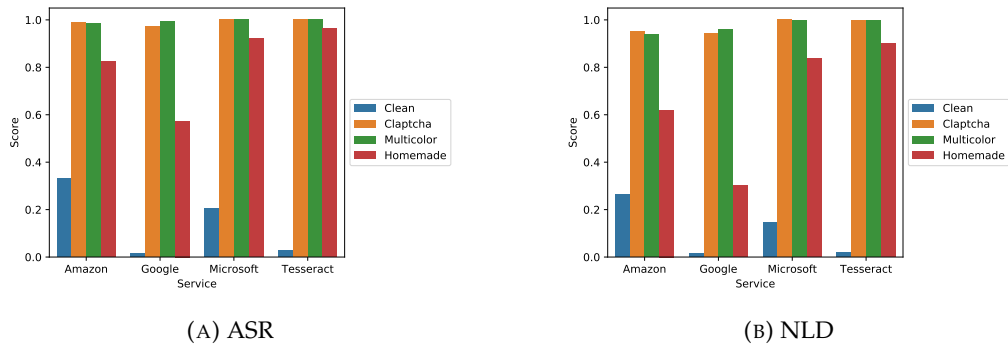


FIGURE 8.12: Cross-domain evaluation. On the left, the Attack Success Rate (ASR). On the right, the average Normalized Levenshtein Distance (NLD). For both measures, the higher, the more successful the attack.

TABLE 8.6: Percentage of D_{tox}^m captchas detected by models trained on data coming from different OSNs.

| Dataset | Pinterest | | | Twitter | | | Yahoo-Flickr | | |
|----------|-----------|----------|----------|---------|----------|----------|--------------|----------|----------|
| | Clap | Multicol | Homemade | Clap | Multicol | Homemade | Clap | Multicol | Homemade |
| Naïve | 11.95 | 0 | 0 | 1.2 | 0 | 0 | 51.79 | 0 | 0.2 |
| Alexnet | 11.16 | 0 | 0 | 1 | 0 | 0 | 52.19 | 0 | 0 |
| Resnet18 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| VGG | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

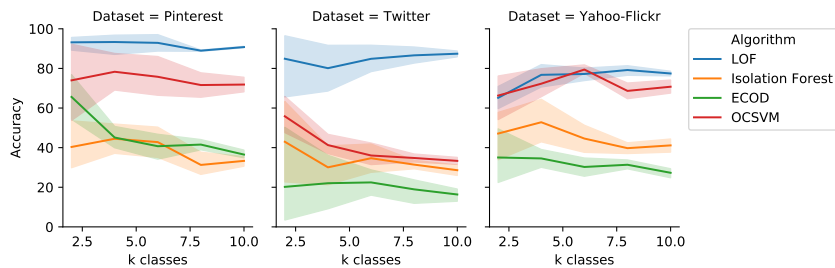


FIGURE 8.13: Accuracy of different Outlier Detection on known captcha styles at the varying of the OSN.

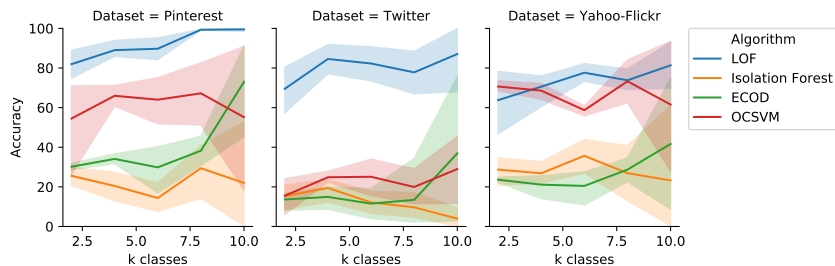


FIGURE 8.14: Accuracy of different Outlier Detection on unknown captcha styles at the varying of the OSN.

Chapter 9

Conclusion and Future Work

The widely adoption of data driven approaches in cybersecurity lead to the advancement of the state-of-the-art in many themes, from the malware to the spam detection. This dissertation aims to present benefits and drawbacks derived by the conjunction of the disciplines of machine learning and cybersecurity. Among the benefits, we saw examples of NIDS such as XeNIDS (in Chapter 2) and DETONAR (in Chapter 3), or applications that analyze web content, from reviews (in Chapter 4) to deepfake videos (in Chapter 5). Among the drawbacks, we saw examples of how attackers can undermined ML algorithms, such as ZeW (in Chapter 7) and CAPA (in Chapter 8).

This dissertation presents many opportunities for future research directions.

- XeNID (Chapter 2): the definition of a new testbed that considers different network topologies while maintaining the same type of attacks is crucial and is fundamental to letting researchers properly validate models cross-networks.
- DETONAR (Chapter 3): DETONAR is a light NIDS that can be easily adopted in the industry. Possible research directions consist in testing such a system in real-life conditions or with a larger number of IoT nodes (currently 16).
- Helpful Reviews prediction with URI (in Chapter 4): this work renews the importance of feature engineering and leads future researchers to investigate more in-depth features derivable by forums organization rather than focusing merely on the ML algorithm choice.
- PRaNA (Chapter 5): the proposed orthogonal approach of analyzing real videos among their deepfake counterparts might open new opportunities to understand the discrepancies between the two categories better.
- ZeW (Chapter 7): the attack shows that evasion attacks can occur by leveraging ML preprocessing algorithms. This work motivates cybersecurity practitioners to investigate better vulnerabilities derivable from ML pipelines rather than focusing on ML algorithms only.
- CAPA (Chapter 8): this study shows that real-life adversarial samples differ from state-of-the-art, being less optimal yet effective. This work highlights the importance of considering real-life attackers when designing ML algorithms and their security.

Bibliography

- [1] ABOU KHAMIS, R., AND MATRAWY, A. Evaluation of Adversarial Training on Different Types of Neural Networks in Deep Learning-based IDSs. In *Proc. IEEE Int. Symp. Netw. Com. Commun.* (2020), pp. 1–6.
- [2] ACHARYA, T., KHATRI, I., ANNAMALAI, A., AND CHOUIKHA, M. F. Efficacy of heterogeneous ensemble assisted machine learning model for binary and multi-class network intrusion detection. In *Proc. IEEE Int. Conf. Aut. Contr. Intell. Syst.* (2021), pp. 408–413.
- [3] AGARWAL, A., XIE, B., VOVSHA, I., RAMBOW, O., AND PASSONNEAU, R. J. Sentiment analysis of twitter data. In *Proceedings of the workshop on language in social media* (2011), pp. 30–38.
- [4] AGIOLLO, A., CONTI, M., KALIYAR, P., LIN, T.-N., AND PAJOLA, L. Detonar: Detection of routing attacks in rpl-based iot. *IEEE Transactions on Network and Service Management* 18, 2 (2021), 1178–1190.
- [5] AHMAD, Z., SHAHID KHAN, A., WAI SHIANG, C., ABDULLAH, J., AND AHMAD, F. Network intrusion detection system: A systematic study of machine learning and deep learning approaches. *T. Emerg. Telecomm. Techn.* 32, 1 (2021), e4150.
- [6] AIREHROUR, D., GUTIERREZ, J. A., AND RAY, S. K. Sectrust-rpl: A secure trust-aware rpl routing protocol for internet of things. *Future Generation Computer Systems* 93 (2019), 860–876.
- [7] ALOQAILY, M., OTOUM, S., RIDHAWI, I. A., AND JARARWEH, Y. An intrusion detection system for connected vehicles in smart cities. *Ad Hoc Networks* 90 (2019), 101842. Recent advances on security and privacy in Intelligent Transportation Systems.
- [8] ALZANTOT, M., SHARMA, Y., ELGOHARY, A., HO, B., SRIVASTAVA, M. B., AND CHANG, K. Generating natural language adversarial examples. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing* (2018), pp. 2890–2896.
- [9] AMERINI, I., ANAGNOSTOPOULOS, A., MAIANO, L., CELSI, L. R., ET AL. Deep learning for multimedia forensics. *Foundations and Trends® in Computer Graphics and Vision* 12, 4 (2021), 309–457.
- [10] ANTONAKAKIS, M., APRIL, T., BAILEY, M., BERNHARD, M., BURSZTEIN, E., COCHRAN, J., DURUMERIC, Z., HALDERMAN, J. A., INVERNIZZI, L., KALLITIS, M., ET AL. Understanding the mirai botnet. In *26th USENIX security symposium (USENIX Security 17)* (2017), pp. 1093–1110.
- [11] APRUZZESE, G., ANDREOLINI, M., MARCHETTI, M., VENTURI, A., AND COLAJANNI, M. Deep reinforcement adversarial learning against botnet evasion attacks. *IEEE T. Netw. Serv. Manag.* 17, 4 (2020).

- [12] APRUZZESE, G., COLAJANNI, M., FERRETTI, L., GUIDO, A., AND MARCHETTI, M. On the effectiveness of machine and deep learning for cybersecurity. In *Proc. IEEE Int. Conf. Cyber Conflicts* (May 2018), pp. 371–390.
- [13] APRUZZESE, G., COLAJANNI, M., FERRETTI, L., AND MARCHETTI, M. Addressing adversarial attacks against security systems based on machine learning. In *Proc. IEEE Int. Conf. Cyber Conflicts* (May 2019), pp. 1–18.
- [14] APRUZZESE, G., MARCHETTI, M., COLAJANNI, M., ZOCCOLI, G. G., AND GUIDO, A. Identifying malicious hosts involved in periodic communications. In *Proc. IEEE Int. Symp. Netw. Comput. Appl.* (Oct. 2017), pp. 1–8.
- [15] APRUZZESE, G., PAJOLA, L., AND CONTI, M. The cross-evaluation of machine learning-based network intrusion detection systems. *IEEE Transactions on Network and Service Management* (2022).
- [16] ARP, D., SPREITZENBARTH, M., HUBNER, M., GASCON, H., RIECK, K., AND SIEMENS, C. Drebin: Effective and explainable detection of android malware in your pocket. In *Ndss* (2014), vol. 14, pp. 23–26.
- [17] ASLAN, Ö. A., AND SAMET, R. A comprehensive review on malware detection approaches. *IEEE Access* 8 (2020), 6249–6271.
- [18] ATENIESE, G., MANCINI, L. V., SPOGNARDI, A., VILLANI, A., VITALI, D., AND FELICI, G. Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers. *Int. J. Secur. Netw.* 10, 3 (sep 2015), 137–150.
- [19] BACCIANELLA, S., ESULI, A., AND SEBASTIANI, F. SentiWordNet 3.0: An enhanced lexical resource for sentiment analysis and opinion mining. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)* (Valletta, Malta, May 2010), European Language Resources Association (ELRA).
- [20] BACHL, M., HARTL, A., FABINI, J., AND ZSEBY, T. Walling up backdoors in intrusion detection systems. In *Proc. ACM Workshop Big Data, Machin. Learn., Artif. Intell. Data Commun. Netw.* (2019), pp. 8–13.
- [21] BANSAL, A., AND MAHAPATRA, S. A comparative analysis of machine learning techniques for botnet detection. In *Proceedings of the 10th International Conference on Security of Information and Networks* (2017), pp. 91–98.
- [22] BARRENO, M., NELSON, B., SEARS, R., JOSEPH, A. D., AND TYGAR, J. D. Can machine learning be secure? In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security* (New York, NY, USA, 2006), ASIACCS '06, Association for Computing Machinery, p. 16–25.
- [23] BENNETT, K. Linguistic steganography: Survey, analysis, and robustness concerns for hiding information in text, 2004.
- [24] BIAZ, S., JI, Y., AND AGRAWAL, P. Impact of sniffer deployment on indoor localization. In *2005 International Conference on Collaborative Computing: Networking, Applications and Worksharing* (2005), IEEE, pp. 10–pp.

- [25] BIGGIO, B., CORONA, I., MAIORCA, D., NELSON, B., ŠRNDIĆ, N., LASKOV, P., GIACINTO, G., AND ROLI, F. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases* (2013), Springer, pp. 387–402.
- [26] BIGGIO, B., FUMERA, G., PILLAI, I., AND ROLI, F. Improving image spam filtering using image text features. In *Proc of the fifth conf on email and anti-spam* (2008).
- [27] BIGGIO, B., FUMERA, G., PILLAI, I., AND ROLI, F. A survey and experimental evaluation of image spam filtering techniques. *Pattern recognition letters* 32, 10 (2011), 1436–1446.
- [28] BIGGIO, B., FUMERA, G., AND ROLI, F. Security evaluation of pattern classifiers under attack. *IEEE T. Knowl. Data. Eng.* 26, 4 (2013), 984–996.
- [29] BIGGIO, B., NELSON, B., AND LASKOV, P. Support vector machines under adversarial label noise. In *Asian conference on machine learning* (2011), PMLR, pp. 97–112.
- [30] BIGGIO, B., NELSON, B., AND LASKOV, P. Poisoning attacks against support vector machines. In *Proceedings of the 29th International Conference on International Conference on Machine Learning* (Madison, WI, USA, 2012), ICML'12, Omnipress, p. 1467–1474.
- [31] BIGGIO, B., AND ROLI, F. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition* 84 (2018), 317–331.
- [32] BILGE, L., BALZAROTTI, D., ROBERTSON, W., KIRDA, E., AND KRUEGEL, C. Disclosure: detecting botnet command and control servers through large-scale netflow analysis. In *Proc. ACM Annual Conf. Comput. Secur. Appl.* (12 2012), pp. 129–138.
- [33] BLEI, D. M., NG, A. Y., AND JORDAN, M. I. Latent dirichlet allocation. *the Journal of machine Learning research* 3 (2003), 993–1022.
- [34] BOSTANI, H., AND SHEIKHAN, M. Hybrid of anomaly-based and specification-based ids for internet of things using unsupervised opf based on mapreduce approach. *Computer Communications* 98 (2017), 52–71.
- [35] BOX, G. E., JENKINS, G. M., REINSEL, G. C., AND LJUNG, G. M. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.
- [36] BREIVOLD, H. P., AND SANDSTRÖM, K. Internet of things for industrial automation—challenges and technical solutions. In *2015 IEEE International Conference on Data Science and Data Intensive Systems* (2015), IEEE, pp. 532–539.
- [37] BRESNIKER, K., GAVRILOVSKA, A., HOLT, J., MILOJICIC, D., AND TRAN, T. Grand challenge: Applying artificial intelligence and machine learning to cybersecurity. *Computer* 52, 12 (2019), 45–52.
- [38] BROWN, S., MILKOV, P., PATEL, S., LOOI, Y. Z., DONG, Z., GU, H., ARTAN, N. S., AND JAIN, E. Acoustic and visual approaches to adversarial text generation for google perspective. In *2019 International Conference on Computational Science and Computational Intelligence (CSCI)* (2019), IEEE, pp. 355–360.

- [39] BUCZAK, A. L., AND GUVEN, E. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications surveys & tutorials* 18, 2 (2015), 1153–1176.
- [40] BUITINCK, L., LOUPPE, G., BLONDEL, M., PEDREGOSA, F., MUELLER, A., GRISEL, O., NICULAE, V., PRETTENHOFER, P., GRAMFORT, A., GROBLER, J., LAYTON, R., VANDERPLAS, J., JOLY, A., HOLT, B., AND VAROQUAUX, G. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning* (2013), pp. 108–122.
- [41] BURSZTEIN, E., AIGRAIN, J., MOSCICKI, A., AND MITCHELL, J. C. The end is nigh: Generic solving of text-based captchas. In *8th {USENIX} Workshop on Offensive Technologies* (2014).
- [42] BURSZTEIN, E., MARTIN, M., AND MITCHELL, J. Text-based CAPTCHA strengths and weaknesses. In *Proc. ACM Conf. Comp. Commun. Secur.* (2011), pp. 125–138.
- [43] CALDELLI, R., GALTERI, L., AMERINI, I., AND DEL BIMBO, A. Optical flow based cnn for detection of unlearned deepfake manipulations. *Pattern Recognition Letters* 146 (2021), 31–37.
- [44] CATILLO, M., DEL VECCHIO, A., OCONE, L., PECCHIA, A., AND VILLANO, U. USB-IDS-1: a Public Multilayer Dataset of Labeled Network Flows for IDS Evaluation. In *Proc. IEEE Int. Conf. Depend. Syst. Netw.* (2021), pp. 1–6.
- [45] CER, D., YANG, Y., KONG, S.-Y., HUA, N., LIMTIACO, N., JOHN, R. S., CONSTANT, N., GUAJARDO-CESPEDES, M., YUAN, S., TAR, C., ET AL. Universal sentence encoder for english. In *Proceedings of the 2018 conference on empirical methods in natural language processing: system demonstrations* (2018), pp. 169–174.
- [46] CERVANTES, C., POPLADE, D., NOGUEIRA, M., AND SANTOS, A. Detection of sinkhole attacks for supporting secure routing on 6lowpan for internet of things. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)* (2015), IEEE, pp. 606–611.
- [47] CHAKRABORTY, A., ALAM, M., DEY, V., CHATTOPADHYAY, A., AND MUKHOPADHYAY, D. Adversarial attacks and defences: A survey. *arXiv preprint arXiv:1810.00069* (2018).
- [48] CHELLAPILLA, K., LARSON, K., SIMARD, P. Y., AND CZERWINSKI, M. Computers beat humans at single character recognition in reading based human interaction proofs. In *CEAS* (2005).
- [49] CHEN, C., YANG, Y., ZHOU, J., LI, X., AND BAO, F. S. Cross-domain review helpfulness prediction based on convolutional neural networks with auxiliary domain discriminators. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)* (New Orleans, Louisiana, June 2018), Association for Computational Linguistics, pp. 602–607.
- [50] CHEN, H., LIU, R., PARK, N., AND SUBRAHMANIAN, V. S. Using twitter to predict when vulnerabilities will be exploited. In *Proceedings of the 25th ACM*

- SIGKDD International Conference on Knowledge Discovery & Data Mining* (2019), p. 3143–3152.
- [51] CHEN, L., SUN, J., AND XU, W. Fawa: fast adversarial watermark attack on optical character recognition (ocr) systems. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (2020), Springer, pp. 547–563.
- [52] CHEN, L., AND XU, W. Attacking optical character recognition (ocr) systems with adversarial watermarks, 2020.
- [53] CHEN, M., FRIDRICH, J., GOLJAN, M., AND LUKAS, J. Determining image origin and integrity using sensor noise. *IEEE Transactions on Information Forensics and Security* 3, 1 (2008), 74–90.
- [54] CHEN, P.-Y., DHANASOBHON, S., AND SMITH, M. D. All reviews are not created equal: The disaggregate impact of reviews and reviewers at amazon.com. *Com* (May 2008) (2008).
- [55] CHEN, W., SU, Y., SHEN, Y., CHEN, Z., YAN, X., AND WANG, W. Y. How large a vocabulary does text classification need? A variational approach to vocabulary selection. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (2019).
- [56] CHENG, M., YI, J., CHEN, P.-Y., ZHANG, H., AND HSIEH, C.-J. Seq2sick: Evaluating the robustness of sequence-to-sequence models with adversarial examples. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2020), vol. 34, pp. 3601–3608.
- [57] CHENG, Y., JIANG, L., AND MACHEREY, W. Robust neural machine translation with doubly adversarial inputs. In *Proceedings of the 57th Conference of the Association for Computational Linguistics* (2019), pp. 4324–4333.
- [58] CHEVALIER, J. A., AND MAYZLIN, D. The effect of word of mouth on sales: Online book reviews. *Journal of marketing research* 43, 3 (2006), 345–354.
- [59] CHUNG, J., GÜLÇEHRE, Ç., CHO, K., AND BENGIO, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014.
- [60] CIFTCI, U. A., DEMIR, I., AND YIN, L. Fakecatcher: Detection of synthetic portrait videos using biological signals. *IEEE transactions on pattern analysis and machine intelligence* (2020).
- [61] CNN, S. L. FDA confirms that St. Jude’s cardiac devices can be hacked. <https://money.cnn.com/2017/01/09/technology/fda-st-jude-cardiac-hack/>.
- [62] COLLOTTA, M., PAU, G., TALTY, T., AND TONGUZ, O. K. Bluetooth 5: A concrete step forward toward the iot. *IEEE Communications Magazine* 56, 7 (2018), 125–131.
- [63] COMPAGNO, A., CONTI, M., LAIN, D., LOVISOTTO, G., AND MANCINI, L. V. Boten elisa: A novel approach for botnet c&c in online social networks. In *2015 IEEE Conference on Communications and Network Security (CNS)* (2015), IEEE, pp. 74–82.

- [64] CONTI, M., KALIYAR, P., AND LAL, C. A robust multicast communication protocol for low power and lossy networks. *Journal of Network and Computer Applications* 164 (2020), 102675.
- [65] CONTI, M., KALIYAR, P., RABBANI, M. M., AND RANISE, S. Split: A secure and scalable rpl routing protocol for internet of things. In *2018 14th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)* (2018), IEEE, pp. 1–8.
- [66] CONTI, M., PAJOLA, L., AND TRICOMI, P. P. Captcha attack: Turning captchas against humanity. *arXiv preprint arXiv:2201.04014* (2022).
- [67] CORDERO, C. G., VASILOMANOLAKIS, E., WAINAKH, A., MÜHLHÄUSER, M., AND NADJM-TEHRANI, S. On generating network traffic datasets with synthetic attacks for intrusion detection. *ACM T. Privacy Secur.* 24, 2 (2021), 1–39.
- [68] COULTER, R., HAN, Q.-L., PAN, L., ZHANG, J., AND XIANG, Y. Data-driven cyber security in perspective—intelligent traffic analysis. *IEEE transactions on cybernetics* 50, 7 (2019), 3081–3093.
- [69] DA XU, L., HE, W., AND LI, S. Internet of things in industries: A survey. *IEEE Transactions on industrial informatics* 10, 4 (2014), 2233–2243.
- [70] DANESCU-NICULESCU-MIZIL, C., KOSSINETS, G., KLEINBERG, J., AND LEE, L. How opinions are received by online communities: A case study on amazon.com helpfulness votes. In *Proceedings of the 18th International Conference on World Wide Web* (New York, NY, USA, 2009), WWW '09, Association for Computing Machinery, p. 141–150.
- [71] DANG, H., LIU, F., STEHOUSER, J., LIU, X., AND JAIN, A. K. On the detection of digital face manipulation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2020), pp. 5780–5789.
- [72] DARKTRACE. Machine Learning in the Age of Cyber AI, 2020.
- [73] DAVIDSON, T., WARMSLEY, D., MACY, M. W., AND WEBER, I. Automated hate speech detection and the problem of offensive language. In *Proceedings of the Eleventh International Conference on Web and Social Media* (2017), pp. 512–515.
- [74] DAVOLI, L., BELLI, L., CILFONE, A., AND FERRARI, G. From micro to macro iot: Challenges and solutions in the integration of ieee 802.15. 4/802.11 and sub-ghz technologies. *IEEE Internet of Things Journal* 5, 2 (2017), 784–793.
- [75] DAYAN, I., ROTH, H. R., ZHONG, A., HAROUNI, A., GENTILI, A., ABIDIN, A. Z., LIU, A., COSTA, A. B., WOOD, B. J., TSAI, C.-S., ET AL. Federated learning for predicting clinical outcomes in patients with COVID-19. *Nature Medicine* (2021), 1–9.
- [76] DE WEEVER, C., WILCZEK, S., DE LAAT, C., AND GERADTS, Z. Deepfake detection through prnu and logistic regression analyses, 2020.
- [77] DEMONTIS, A., MELIS, M., PINTOR, M., JAGIELSKI, M., BIGGIO, B., OPREA, A., NITA-ROTARU, C., AND ROLI, F. Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks. In *28th {USENIX} Security Symposium ({USENIX} Security 19)* (2019), pp. 321–338.

- [78] DIAZ, G. O., AND NG, V. Modeling and prediction of online product review helpfulness: a survey. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (Melbourne, Australia, 2018), Association for Computational Linguistics, pp. 698–708.
- [79] DIVEKAR, A., PAREKH, M., SAVLA, V., MISHRA, R., AND SHIROLE, M. Benchmarking datasets for anomaly-based network intrusion detection: Kdd cup 99 alternatives. In *Proc. IEEE Int. Conf. Comp. Commun. Secur.* (2018), pp. 1–8.
- [80] DOLHANSKY, B., BITTON, J., PFLAUM, B., LU, J., HOWES, R., WANG, M., AND FERRER, C. C. The deepfake detection challenge (dfdc) dataset. *arXiv preprint arXiv:2006.07397* (2020).
- [81] DORIGUZZI-CORIN, R., MILLAR, S., SCOTT-HAYWARD, S., MARTINEZ-DEL RINCON, J., AND SIRACUSA, D. LUCID: A practical, lightweight deep learning solution for DDoS attack detection. *IEEE Transactions on Network and Service Management* 17, 2 (2020), 876–889.
- [82] DU, J., RONG, J., MICHALSKA, S., WANG, H., AND ZHANG, Y. Feature selection for helpfulness prediction of online product reviews: An empirical study. *PloS one* 14, 12 (2019), e0226902.
- [83] DUAN, W., GU, B., AND WHINSTON, A. B. The dynamics of online word-of-mouth and product sales—an empirical investigation of the movie industry. *Journal of retailing* 84, 2 (2008), 233–242.
- [84] DUNN, C., MOUSTAFA, N., AND TURNBULL, B. Robustness evaluations of sustainable machine learning models against data poisoning attacks in the internet of things. *Sustainability* 12, 16 (2020), 6434.
- [85] D’ANGELO, G., FICCO, M., AND PALMIERI, F. Malware detection in mobile environments based on Autoencoders and API-images. *Elsevier J. Parallel Distrib. Comp.* 137 (2020), 26–33.
- [86] ESLAMI, S. P., GHASEMAGHAEI, M., AND HASSANEIN, K. Which online reviews do consumers find most helpful? a multi-method investigation. *Decision Support Systems* 113 (2018), 32–42.
- [87] ESTEVA, A., ROBICQUET, A., RAMSUNDAR, B., KULESHOV, V., DEPRISTO, M., CHOU, K., CUI, C., CORRADO, G., THRUN, S., AND DEAN, J. A guide to deep learning in healthcare. *Nature medicine* 25, 1 (2019), 24–29.
- [88] FALCÃO, F., ZOPPI, T., SILVA, C. B. V., SANTOS, A., FONSECA, B., CECCARELLI, A., AND BONDAVALLI, A. Quantitative comparison of unsupervised anomaly detection algorithms for intrusion detection. In *Proc. ACM Symp. Appl. Comp.* (2019), pp. 318–327.
- [89] FAN, M., FENG, C., GUO, L., SUN, M., AND LI, P. Product-aware helpfulness prediction of online reviews. In *The World Wide Web Conference* (New York, NY, USA, 2019), WWW ’19, Association for Computing Machinery, p. 2715–2721.
- [90] FENG, X., LIAO, X., WANG, X., WANG, H., LI, Q., YANG, K., ZHU, H., AND SUN, L. Understanding and securing device vulnerabilities through automated bug report analysis. In *SEC’19: Proceedings of the 28th USENIX Conference on Security Symposium* (2019).

- [91] FILIERI, R. What makes online reviews helpful? a diagnosticity-adoption framework to explain informational and normative influences in e-wom. *Journal of business research* 68, 6 (2015), 1261–1270.
- [92] FLESHMAN, W., RAFF, E., ZAK, R., MCLEAN, M., AND NICHOLAS, C. Static malware detection & subterfuge: Quantifying the robustness of machine learning and current anti-virus. In *Proc. IEEE Int. Conf. Malicious Unwanted Soft.* (2018), pp. 1–10.
- [93] FREDRIKSON, M., JHA, S., AND RISTENPART, T. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security* (2015), pp. 1322–1333.
- [94] FREDRIKSON, M., LANTZ, E., JHA, S., LIN, S., PAGE, D., AND RISTENPART, T. Privacy in pharmacogenetics: An {End-to-End} case study of personalized warfarin dosing. In *23rd USENIX Security Symposium (USENIX Security 14)* (2014), pp. 17–32.
- [95] GAIKWAD, P. P., GABHANE, J. P., AND GOLAIT, S. S. A survey based on smart homes system using internet-of-things. In *2015 International Conference on Computation of Power, Energy, Information and Communication (ICCPEIC)* (2015), IEEE, pp. 0330–0335.
- [96] GAMA, J., ŽLIOBAITĚ, I., BIFET, A., PECHENIZKIY, M., AND BOUCHACHIA, A. A survey on concept drift adaptation. *ACM computing surveys (CSUR)* 46, 4 (2014), 1–37.
- [97] GAO, H., YAN, J., CAO, F., ZHANG, Z., LEI, L., TANG, M., ZHANG, P., ZHOU, X., WANG, X., AND LI, J. A simple generic attack on text captchas. In *NDSS* (2016).
- [98] GAO, J., LANCHANTIN, J., SOFFA, M. L., AND QI, Y. Black-box generation of adversarial text sequences to evade deep learning classifiers. In *2018 IEEE Security and Privacy Workshops (SPW)* (2018), IEEE, pp. 50–56.
- [99] GARA, F., SAAD, L. B., AND AYED, R. B. An intrusion detection system for selective forwarding attack in ipv6-based mobile wsns. In *2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)* (2017), IEEE, pp. 276–281.
- [100] GARA, F., SAAD, L. B., AYED, R. B., AND TOURANCHEAU, B. Rpl protocol adapted for healthcare and medical applications. In *2015 International wireless communications and mobile computing conference (IWCMC)* (2015), IEEE, pp. 690–695.
- [101] GARCIA, S., GRILL, M., STIBOREK, J., AND ZUNINO, A. An empirical comparison of botnet detection methods. *computers & security* 45 (2014), 100–123.
- [102] GHOSE, A., AND IPEIROTIS, P. G. Estimating the helpfulness and economic impact of product reviews: Mining text and reviewer characteristics. *IEEE transactions on knowledge and data engineering* 23, 10 (2011), 1498–1512.
- [103] GILLESPIE, T. Content moderation, ai, and the question of scale. *Big Data & Society* 7, 2 (2020), 2053951720943234.

- [104] GO, A., BHAYANI, R., AND HUANG, L. Twitter sentiment classification using distant supervision. *CS224N project report, Stanford 1*, 12 (2009), 2009.
- [105] GODES, D., AND SILVA, J. C. Sequential and temporal dynamics of online opinion. *Marketing Science* 31, 3 (2012), 448–473.
- [106] GOLJAN, M., FRIDRICH, J., AND FILLER, T. Large scale test of sensor fingerprint camera identification. In *Media forensics and security* (2009), vol. 7254, International Society for Optics and Photonics, p. 72540I.
- [107] GOMEZ, J. C. pinterest_dataset v3. *10.17632/fs4k2zc5j5.3* (2018).
- [108] GOMEZ, R., GIBERT, J., GOMEZ, L., AND KARATZAS, D. Exploring hate speech detection in multimodal publications. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision* (2020), pp. 1470–1478.
- [109] GONG, Z., WANG, W., LI, B., SONG, D., AND KU, W.-S. Adversarial texts with gradient methods, 2018. Withdrawn.
- [110] GOODFELLOW, I., POUGET-ABADIE, J., MIRZA, M., XU, B., WARDE-FARLEY, D., OZAI, S., COURVILLE, A., AND BENGIO, Y. Generative adversarial nets. In *Advances in Neural Information Processing Systems* (2014), pp. 2672–2680.
- [111] GOODFELLOW, I., SHLENS, J., AND SZEGEDY, C. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations* (2015).
- [112] GOODFELLOW, I. J., BULATOV, Y., IBARZ, J., ARNOUD, S., AND SHET, V. Multi-digit number recognition from street view imagery using deep convolutional neural networks. *arXiv preprint arXiv:1312.6082* (2013).
- [113] GOODFELLOW, I. J., SHLENS, J., AND SZEGEDY, C. Explaining and harnessing adversarial examples, 2015.
- [114] GRIGORESCU, S., TRASNEA, B., COCIAS, T., AND MACESANU, G. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics* 37, 3 (2020), 362–386.
- [115] GRÖNDAHL, T., PAJOLA, L., JUUTI, M., CONTI, M., AND ASOKAN, N. All you need is" love" evading hate speech detection. In *Proceedings of the 11th ACM workshop on artificial intelligence and security* (2018), pp. 2–12.
- [116] GROSSE, K., BIERINGER, L., BESOLD, T. R., BIGGIO, B., AND KROMBHOZ, K. " why do so?"—a practical perspective on machine learning security. *arXiv preprint arXiv:2207.05164* (2022).
- [117] GU, T., DOLAN-GAVITT, B., AND GARG, S. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733* (2017).
- [118] GUERAR, M., VERDERAME, L., MIGLIARDI, M., PALMIERI, F., AND MERLO, A. Gotta captcha'em all: A survey of twenty years of the human-or-computer dilemma. *arXiv preprint arXiv:2103.01748* (2021).
- [119] GÜERA, D., AND DELP, E. J. Deepfake video detection using recurrent neural networks. In *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)* (2018), pp. 1–6.

- [120] HATEBASE. Hatebase. hatebase.org/ (2021). Accessed on 2021-03-22.
- [121] HAVAEI, M., DAVY, A., WARDE-FARLEY, D., BIARD, A., COURVILLE, A., BENGIO, Y., PAL, C., JODOIN, P.-M., AND LAROCHELLE, H. Brain tumor segmentation with deep neural networks. *Medical image analysis* 35 (2017), 18–31.
- [122] HE, R., AND MCAULEY, J. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *Proceedings of the 25th International Conference on World Wide Web* (2016), International World Wide Web Conferences Steering Committee, p. 507–517.
- [123] HOCHREITER, S., AND SCHMIDHUBER, J. Long short-term memory. *Neural Computation* 9, 8 (1997), 1735–1780.
- [124] HODO, E., BELLEKENS, X., HAMILTON, A., DUBOUILH, P.-L., IORKYASE, E., TACHTATZIS, C., AND ATKINSON, R. Threat analysis of iot networks using artificial neural network intrusion detection system. In *2016 International Symposium on Networks, Computers and Communications (ISNCC)* (2016), IEEE, pp. 1–6.
- [125] HONG, Y., LU, J., YAO, J., ZHU, Q., AND ZHOU, G. What reviews are satisfactory: Novel features for automatic helpfulness voting. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval* (New York, NY, USA, 2012), SIGIR '12, Association for Computing Machinery, p. 495–504.
- [126] HORÁK, M., STUPKA, V., AND HUSÁK, M. GDPR compliance in cybersecurity software: a case study of DPIA in information sharing platform. In *Proc. ACM Int. Conf. Availab. Reliab. Secur.* (2019), pp. 1–8.
- [127] HOSSEINI, H., KANNAN, S., ZHANG, B., AND POOVENDRAN, R. Deceiving google’s perspective API built for detecting toxic comments, 2017.
- [128] HOU, J., FU, P., CAO, Z., AND XU, A. Machine learning based DDoS detection through NetFlow analysis. In *Proc. IEEE Milit. Commun. Conf.* (2018), pp. 1–6.
- [129] HOWARD, J., AND RUDER, S. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (Melbourne, Australia, July 2018), Association for Computational Linguistics, pp. 328–339.
- [130] HU, M., AND LIU, B. Mining and summarizing customer reviews. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2004), KDD '04, Association for Computing Machinery, p. 168–177.
- [131] HUANG, A. H., CHEN, K., YEN, D. C., AND TRAN, T. P. A study of factors that contribute to online review helpfulness. *Computers in Human Behavior* 48 (2015), 17–27.
- [132] HUANG, L., JOSEPH, A. D., NELSON, B., RUBINSTEIN, B. I., AND TYGAR, J. D. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence* (2011), pp. 43–58.

- [133] HUTTO, C., AND GILBERT, E. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Proceedings of the International AAAI Conference on Web and Social Media* (2014), vol. 8.
- [134] HYNDMAN, R. J., AND KHANDAKAR, Y. Automatic time series forecasting: the forecast package for r. *Journal of statistical software* 27 (2008), 1–22.
- [135] INJADAT, M., MOUBAYED, A., NASSIF, A. B., AND SHAMI, A. Multi-stage optimized machine learning framework for network intrusion detection. *IEEE T. Netw. Serv. Manag.* (2020).
- [136] INKAWHICH, N. Finetuning torchvision models. pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html (2021). Accessed on 2021-09-10.
- [137] INSTAGRAM. Introducing sensitive content control. about.fb.com/news/2021/07/introducing-sensitive-content-control (2021). Accessed on 2021-08-10.
- [138] INTELLIGENCE, M. Machine learning as a service (mlaas) market - growth, trends, and forecast (2020 - 2025). <https://www.mordorintelligence.com/industry-reports/global-machine-learning-as-a-service-mlaas-market> (2019).
- [139] ISLAM, S. R., KWAK, D., KABIR, M. H., HOSSAIN, M., AND KWAK, K.-S. The internet of things for health care: a comprehensive survey. *IEEE access* 3 (2015), 678–708.
- [140] JI, Y., BIAZ, S., WU, S., AND QI, B. Optimal sniffers deployment on wireless indoor localization. In *2007 16th International Conference on Computer Communications and Networks* (2007), IEEE, pp. 251–256.
- [141] JIA, X., FENG, Q., FAN, T., AND LEI, Q. Rfid technology and its applications in internet of things (iot). In *2012 2nd international conference on consumer electronics, communications and networks (CECNet)* (2012), IEEE, pp. 1282–1285.
- [142] JINDAL, N., AND LIU, B. Opinion spam and analysis. In *Proceedings of the 2008 International Conference on Web Search and Data Mining* (New York, NY, USA, 2008), WSDM '08, Association for Computing Machinery, p. 219–230.
- [143] JORDANEY, R., SHARAD, K., DASH, S. K., WANG, Z., PAPINI, D., NOURETDINOV, I., AND CAVALLARO, L. TRANSCEND: Detecting concept drift in malware classification models. In *Proc. USENIX Secur. Symp.* (2017), pp. 625–642.
- [144] JOYCE, R. J., RAFF, E., AND NICHOLAS, C. A framework for cluster and classifier evaluation in the absence of reference labels. In *Proc. ACM Comp. Commun. Secur. AI Sec Workshop* (2021).
- [145] JUNG, W., HONG, S., HA, M., KIM, Y.-J., AND KIM, D. Ssl-based lightweight security of ip-based wireless sensor networks. In *2009 International Conference on Advanced Information Networking and Applications Workshops* (2009), IEEE, pp. 1112–1117.

- [146] JUUTI, M., SUN, B., MORI, T., AND ASOKAN, N. Stay on-topic: Generating context-specific fake restaurant reviews. In *European Symposium on Research in Computer Security* (Cham, 2018), Springer, Springer International Publishing, pp. 132–151.
- [147] KALIYAR, P., JABALLAH, W. B., CONTI, M., AND LAL, C. Lidl: localization with early detection of sybil and wormhole attacks in iot networks. *Computers & Security* 94 (2020), 101849.
- [148] KAPADIA, S. Topic modeling in python: Latent dirichlet allocation (lda), 2019.
- [149] KARRAS, T., LAINE, S., AND AILA, T. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2019), pp. 4401–4410.
- [150] KHALID, U., ASIM, M., BAKER, T., HUNG, P. C., TARIQ, M. A., AND RAF-FERTY, L. A decentralized lightweight blockchain-based authentication mechanism for iot systems. *Cluster Computing* 23, 3 (2020), 2067–2087.
- [151] KHARRUFA, H., AL-KASHOASH, H. A., AND KEMP, A. H. Rpl-based routing protocols in iot applications: a review. *IEEE Sensors Journal* 19, 15 (2019), 5952–5967.
- [152] KHODABAKHSH, A., RAMACHANDRA, R., RAJA, K., WASNIK, P., AND BUSCH, C. Fake face detection methods: Can they be generalized? In *IEEE International Conference of the Biometrics Special Interest Group* (2018), pp. 1–6.
- [153] KIELA, D., FIROOZ, H., MOHAN, A., GOSWAMI, V., SINGH, A., RINGSHIA, P., AND TESTUGGINE, D. The hateful memes challenge: Detecting hate speech in multimodal memes. *Advances in Neural Information Processing Systems* 33 (2020), 2611–2624.
- [154] KIM, H.-S., KO, J., CULLER, D. E., AND PAEK, J. Challenging the ipv6 routing protocol for low-power and lossy networks (rpl): A survey. *IEEE Communications Surveys & Tutorials* 19, 4 (2017), 2502–2525.
- [155] KIM, H.-S., PAEK, J., AND BAHK, S. Qu-rpl: Queue utilization based rpl for load balancing in large scale industrial applications. In *2015 12th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)* (2015), IEEE, pp. 265–273.
- [156] KIM, S.-M., PANTEL, P., CHKLOVSKI, T., AND PENNACCHIOTTI, M. Automatically assessing review helpfulness. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing* (USA, 2006), EMNLP '06, Association for Computational Linguistics, p. 423–430.
- [157] KLOFT, M., AND LASKOV, P. Online anomaly detection under adversarial impact. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010), Y. W. Teh and M. Titterington, Eds., vol. 9 of *Proceedings of Machine Learning Research*, PMLR, pp. 405–412.
- [158] KOLIAS, C., KAMBOURAKIS, G., STAVROU, A., AND VOAS, J. Ddos in the iot: Mirai and other botnets. *Computer* 50, 7 (2017), 80–84.

- [159] KOOPMAN, M., RODRIGUEZ, A. M., AND GERADTS, Z. Detection of deepfake video manipulation. In *The 20th Irish machine vision and image processing conference (IMVIP)* (2018), pp. 133–136.
- [160] KORDE, V. Text classification and classifiers: A survey. *International Journal of Artificial Intelligence & Applications* 3 (2012), 85–99.
- [161] KORFIATIS, N., GARCÍA-BARIOCANAL, E., AND SÁNCHEZ-ALONSO, S. Evaluating content quality and helpfulness of online product reviews: The interplay of review helpfulness vs. review content. *Electronic Commerce Research and Applications* 11, 3 (2012), 205–217.
- [162] KOUOKAM, E. K., AND DIRIK, A. E. Prnu-based source device attribution for youtube videos. *Digital Investigation* 29 (2019), 91–100.
- [163] KRIZHEVSKY, A., SUTSKEVER, I., AND HINTON, G. E. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* 25 (2012), 1097–1105.
- [164] KROL, K., PARKIN, S., AND SASSE, M. A. Better the devil you know: A user study of two captchas and a possible replacement technology. In *NDSS Workshop on Usable Security* (2016), vol. 10.
- [165] KRUEGEL, C., AND VIGNA, G. Anomaly detection of web-based attacks. In *Proceedings of the 10th ACM conference on Computer and communications security* (2003), Association for Computing Machinery, p. 251–261.
- [166] KUSNER, M. J., SUN, Y., KOLKIN, N. I., AND WEINBERGER, K. Q. From word embeddings to document distances. In *Proceedings of the 32nd International Conference on Machine Learning* (2015), pp. 957–966.
- [167] LASTLINE. Using AI to detect and contain Cyberthreats, 2019.
- [168] LEE, S., AND CHOEH, J. Y. Predicting the helpfulness of online reviews using multilayer perceptron neural networks. *Expert Systems with Applications* 41, 6 (2014), 3041–3046.
- [169] LI, J., JI, S., DU, T., LI, B., AND WANG, T. Textbugger: Generating adversarial text against real-world applications. In *26th Annual Network and Distributed System Security Symposium* (2019).
- [170] LI, L., BAO, J., ZHANG, T., YANG, H., CHEN, D., WEN, F., AND GUO, B. Face x-ray for more general face forgery detection. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2020), pp. 5000–5009.
- [171] LI, L., XIAO GUANG, H., KE, C., AND KETAI, H. The applications of wifi-based wireless sensor network in internet of things and smart grid. In *2011 6th IEEE Conference on Industrial Electronics and Applications* (2011), IEEE, pp. 789–793.
- [172] LI, Y., CHANG, M.-C., AND LYU, S. In icu oculi: Exposing ai created fake videos by detecting eye blinking. In *2018 IEEE International workshop on information forensics and security (WIFS)* (2018), IEEE, pp. 1–7.
- [173] LI, Y., AND LYU, S. Exposing deepfake videos by detecting face warping artifacts. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (2019).

- [174] LI, Y., YANG, X., SUN, P., QI, H., AND LYU, S. Celeb-df: A large-scale challenging dataset for deepfake forensics. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2020), pp. 3204–3213.
- [175] LI, Z., ZHAO, Y., HU, X., BOTTA, N., IONESCU, C., AND CHEN, G. H. Ecod: Unsupervised outlier detection using empirical cumulative distribution functions. *arXiv preprint arXiv:2201.00382* (2022).
- [176] LIANG, B., SU, M., YOU, W., SHI, W., AND YANG, G. Cracking classifiers for evasion: a case study on the google’s phishing pages filter. In *Proceedings of the 25th International Conference on World Wide Web* (2016), pp. 345–356.
- [177] LIAO, H.-J., LIN, C.-H. R., LIN, Y.-C., AND TUNG, K.-Y. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications* 36, 1 (2013), 16–24.
- [178] LIM, S. C. J., LIU, Y., AND LEE, W. B. Faceted search and retrieval based on semantically annotated product family ontology. In *Proceedings of the WSDM ’09 Workshop on Exploiting Semantic Annotations in Information Retrieval* (New York, NY, USA, 2009), ESAIR ’09, Association for Computing Machinery, p. 15–24.
- [179] LIN, H., YANG, P., AND ZHANG, F. Review of scene text detection and recognition. *Archives of computational methods in engineering* 27, 2 (2020), 433–454.
- [180] LIU, B. *Sentiment Analysis and Opinion Mining*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2012.
- [181] LIU, J., CAO, Y., LIN, C.-Y., HUANG, Y., AND ZHOU, M. Low-quality product review detection in opinion summarization. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)* (Prague, Czech Republic, June 2007), Association for Computational Linguistics, pp. 334–342.
- [182] LIU, Y., CHEN, X., LIU, C., AND SONG, D. Delving into transferable adversarial examples and black-box attacks. In *Proceedings of 5th International Conference on Learning Representations* (2017).
- [183] LIU, Y., HUANG, X., AN, A., AND YU, X. Modeling and predicting the helpfulness of online reviews. In *IEEE International Conference on Data Mining* (2008), pp. 443–452.
- [184] LIU, Y., JIN, J., JI, P., HARDING, J. A., AND FUNG, R. Y. Identifying helpful online reviews: A product designer’s perspective. *Computer-Aided Design* 45, 2 (2013), 180–194. Solid and Physical Modeling 2012.
- [185] LONG, S., HE, X., AND YAO, C. Scene text detection and recognition: The deep learning era. *International Journal of Computer Vision* 129, 1 (2021), 161–184.
- [186] LOPER, E., AND BIRD, S. Nltk: The natural language toolkit. *CoRR cs.CL/0205028* (2002).
- [187] LU, Y., TSAPARAS, P., NTOULAS, A., AND POLANYI, L. Exploiting social context for review quality prediction. In *Proceedings of the 19th International Conference on World Wide Web* (New York, NY, USA, 2010), WWW ’10, Association for Computing Machinery, p. 691–700.

- [188] LUGSTEIN, F., BAIER, S., BACHINGER, G., AND UHL, A. Prnu-based deepfake detection. In *Proceedings of the 2021 ACM Workshop on Information Hiding and Multimedia Security* (2021), pp. 7–12.
- [189] LUKÁŠ, J., FRIDRICH, J., AND GOLJAN, M. Detecting digital image forgeries using sensor pattern noise. In *Security, Steganography, and Watermarking of Multimedia Contents VIII* (2006).
- [190] LUONG, T., SUTSKEVER, I., LE, Q. V., VINYALS, O., AND ZAREMBA, W. Addressing the rare word problem in neural machine translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing* (2015), pp. 11–19.
- [191] MAAS, A., DALY, R. E., PHAM, P. T., HUANG, D., NG, A. Y., AND POTTS, C. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies* (2011), pp. 142–150.
- [192] MACAVANEY, S., YAO, H.-R., YANG, E., RUSSELL, K., GOHARIAN, N., AND FRIEDER, O. Hate speech detection: Challenges and solutions. *PloS one* 14, 8 (2019), e0221152.
- [193] MAGÁN-CARRIÓN, R., URDA, D., DÍAZ-CANO, I., AND DORRONSORO, B. Towards a reliable comparison and evaluation of network intrusion detection systems based on machine learning approaches. *Applied Sciences* 10, 5 (2020), 1775.
- [194] MAINETTI, L., PATRONO, L., AND VILEI, A. Evolution of wireless sensor networks towards the internet of things: A survey. In *SoftCOM 2011, 19th international conference on software, telecommunications and computer networks* (2011), pp. 1–6.
- [195] MARRA, F., GRAGNANIELLO, D., COZZOLINO, D., AND VERDOLIVA, L. Detection of gan-generated fake images over social networks. In *2018 IEEE Conference on Multimedia Information Processing and Retrieval* (2018), pp. 384–389.
- [196] MARTIN, L., AND PU, P. Prediction of helpful reviews using emotions extraction. *Proceedings of the AAAI Conference on Artificial Intelligence* 28, 1 (Jun. 2014).
- [197] MATERN, F., RIESS, C., AND STAMMINGER, M. Exploiting visual artifacts to expose deepfakes and face manipulations. In *2019 IEEE Winter Applications of Computer Vision Workshops (WACVW)* (2019), IEEE, pp. 83–92.
- [198] MAYER, F., AND STEINEBACH, M. Forensic image inspection assisted by deep learning. In *Proceedings of the 12th International Conference on Availability, Reliability and Security* (2017), pp. 1–9.
- [199] MAYZAUD, A., BADONNEL, R., AND CHRISMENT, I. A distributed monitoring strategy for detecting version number attacks in rpl-based networks. *IEEE Transactions on Network and Service Management* 14, 2 (2017), 472–486.
- [200] MEMON, J., SAMI, M., KHAN, R. A., AND UDDIN, M. Handwritten optical character recognition (ocr): A comprehensive systematic literature review (slr). *IEEE Access* 8 (2020), 142642–142668.

- [201] MIKOLOV, T., CHEN, K., CORRADO, G., AND DEAN, J. Efficient estimation of word representations in vector space. In *1st International Conference on Learning Representations* (2013).
- [202] MIKOLOV, T., SUTSKEVER, I., CHEN, K., CORRADO, G., AND DEAN, J. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2* (Red Hook, NY, USA, 2013), NIPS'13, Curran Associates Inc., p. 3111–3119.
- [203] MILLER, B., KANTCHELIAN, A., TSCHANTZ, M. C., AFROZ, S., BACHWANI, R., FAIZULLABHOY, R., HUANG, L., SHANKAR, V., WU, T., YIU, G., ET AL. Reviewer integration and performance measurement for malware detection. In *Proc. Int. Conf. DIMVA* (2016), pp. 122–141.
- [204] MIN, E., LONG, J., LIU, Q., CUI, J., CAI, Z., AND MA, J. Su-ids: A semi-supervised and unsupervised framework for network intrusion detection. In *Proc. Springer Int. Conf. Cloud Comp. Secur.* (2018), pp. 322–334.
- [205] MIN, H.-J., AND PARK, J. C. Identifying helpful reviews based on customer's mentions about experiences. *Expert Systems with Applications* 39, 15 (2012), 11830–11838.
- [206] MISHRA, P., VARADHARAJAN, V., TUPAKULA, U., AND PILLI, E. S. A detailed investigation and analysis of using machine learning techniques for intrusion detection. *IEEE Comm. Surv. Tut.* 21, 1 (2018), 686–728.
- [207] MOOSAVI-DEZFOOLI, S., FAWZI, A., AND FROSSARD, P. Deepfool: A simple and accurate method to fool deep neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 2574–2582.
- [208] MORI, G., AND MALIK, J. Recognizing objects in adversarial clutter: Breaking a visual captcha. In *Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.* (2003), vol. 1, IEEE, pp. I–I.
- [209] MOTOYAMA, M., MCCOY, D., LEVCHENKO, K., SAVAGE, S., AND VOELKER, G. M. Dirty jobs: The role of freelance labor in web service abuse. In *20th USENIX Security Symposium (USENIX Security 11)* (San Francisco, CA, Aug. 2011), USENIX Association.
- [2010] MOUSTAFA, N., AND SLAY, J. Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *2015 military communications and information systems conference (MilCIS)* (2015), IEEE, pp. 1–6.
- [2011] MUDAMBI, S. M., AND SCHUFF, D. Research note: What makes a helpful online review? a study of customer reviews on amazon. com. *MIS quarterly* (2010), 185–200.
- [2012] NATHANIEL, Y. Z-wasp vulnerability used to phish office 365 and atp. <https://www.avanan.com/blog/zwasp-microsoft-office-365-phishing-vulnerability> (2019).
- [2013] NGO-YE, T. L., AND SINHA, A. P. The influence of reviewer engagement characteristics on online review helpfulness: A text regression model. *Decision Support Systems* 61 (2014), 47–58.

- [214] NGUYEN, H. H., YAMAGISHI, J., AND ECHIZEN, I. Capsule-forensics: Using capsule networks to detect forged images and videos. In *IEEE International Conference on Acoustics, Speech and Signal Processing* (2019), pp. 2307–2311.
- [215] NGUYEN, T. D., RIEGER, P., MIETTINEN, M., AND SADEGHI, A.-R. Poisoning attacks on federated learning-based IoT intrusion detection system. In *Proc. Workshop Decentralized IoT Syst. Secur.* (2020), pp. 1–7.
- [216] NIWATTANAKUL, S., SINGTHONGCHAI, J., NAENUDORN, E., AND WANAPU, S. Using of jaccard coefficient for keywords similarity. In *Proceedings of the international multiconference of engineers and computer scientists* (2013), vol. 1, pp. 380–384.
- [217] OLIVER, A., ODENA, A., RAFFEL, C. A., CUBUK, E. D., AND GOODFELLOW, I. Realistic Evaluation of Deep Semi-Supervised Learning Algorithms. *Proc. Adv. Neur. Inf. Process. Syst.* 31 (2018), 3235–3246.
- [218] O’MAHONY, M. P., AND SMYTH, B. Learning to recommend helpful hotel reviews. In *Proceedings of the Third ACM Conference on Recommender Systems* (New York, NY, USA, 2009), RecSys ’09, Association for Computing Machinery, p. 305–308.
- [219] O’MAHONY, M. P., AND SMYTH, B. Using readability tests to predict helpful product reviews. In *Adaptivity, Personalization and Fusion of Heterogeneous Information* (Paris, FRA, 2010), RIAO ’10, LE CENTRE DE HAUTES ETUDES INTERNATIONALES D’INFORMATIQUE DOCUMENTAIRE, p. 164–167.
- [220] PAJOLA, L., AND CONTI, M. Fall of giants: How popular text-based mlaas fall against a simple evasion attack. In *2021 IEEE European Symposium on Security and Privacy (EuroS&P)* (2021), IEEE, pp. 198–211.
- [221] PAJOLA, L., PASA, L., AND CONTI, M. Threat is in the air: Machine learning for wireless network applications. In *Proceedings of the ACM Workshop on Wireless Security and Machine Learning* (2019), pp. 16–21.
- [222] PAK, A., AND PAROUBEK, P. Twitter as a corpus for sentiment analysis and opinion mining. In *Proceedings of the International Conference on Language Resources and Evaluation* (2010).
- [223] PAN, Y., AND ZHANG, J. Q. Born unequal: a study of the helpfulness of user-generated product reviews. *Journal of retailing* 87, 4 (2011), 598–612.
- [224] PANG, B., AND LEE, L. Opinion mining and sentiment analysis. *Found. Trends Inf. Retr.* 2, 1-2 (2008), 1–135.
- [225] PAPERNOT, N., MCDANIEL, P. D., AND GOODFELLOW, I. J. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples, 2016.
- [226] PASQUINI, C., AMERINI, I., AND BOATO, G. Media forensics on social media platforms: a survey. *EURASIP Journal on Information Security* 2021, 1 (2021), 1–19.
- [227] PASZKE, A., GROSS, S., MASSA, F., LERER, A., BRADBURY, J., CHANAN, G., KILLEEN, T., LIN, Z., GIMELSHEIN, N., ANTIGA, L., DESMAISON, A., KOPF,

- A., YANG, E., DEVITO, Z., RAISON, M., TEJANI, A., CHILAMKURTHY, S., STEINER, B., FANG, L., BAI, J., AND CHINTALA, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.
- [228] PENNEBAKER, J. W., BOYD, R. L., JORDAN, K., AND BLACKBURN, K. The development and psychometric properties of liwc2015, 2015.
- [229] PENNINGTON, J., SOCHER, R., AND MANNING, C. D. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (Doha, Qatar, 2014), Association for Computational Linguistics, pp. 1532–1543.
- [230] PEREA, M., DUÑABEITIA, J. A., AND CARREIRAS, M. R34d1ng w0rd5 w1th numb3r5. *Journal of Experimental Psychology: Human Perception and Performance* 34 (2008), 237–241.
- [231] PIERAZZI, F., APRUZZESE, G., COLAJANNI, M., GUIDO, A., AND MARCHETTI, M. Scalable architecture for online prioritisation of cyber threats. In *Proc. IEEE Int. Conf. Cyber Conflicts* (May 2017), pp. 1–18.
- [232] PONGLE, P., AND CHAVAN, G. Real time intrusion and wormhole attack detection in internet of things. *International Journal of Computer Applications* 121, 9 (2015).
- [233] PONGLE, P., AND CHAVAN, G. A survey: Attacks on rpl and 6lowpan in iot. In *2015 International conference on pervasive computing (ICPC)* (2015), IEEE, pp. 1–6.
- [234] PONTES, C. F. T., DE SOUZA, M. M. C., GONDIM, J. J. C., BISHOP, M., AND MAROTTA, M. A. A new method for flow-based network intrusion detection using the inverse potts model. *IEEE Transactions on Network and Service Management* 18, 2 (2021), 1125–1136.
- [235] PULSE, M. Amazon number of prime members, 2022.
- [236] QIU, L., PANG, J., AND LIM, K. H. Effects of conflicting aggregated rating on ewom review credibility and diagnosticity: The moderating role of review valence. *Decision Support Systems* 54, 1 (2012), 631–643.
- [237] QU, X., LI, X., FARKAS, C., AND ROSE, J. An attention model of customer expectation to improve review helpfulness prediction. *Advances in information retrieval* 12035 (2020), 836.
- [238] QU, X., LI, Z., WANG, J., ZHANG, Z., ZOU, P., JIANG, J., HUANG, J., XIAO, R., ZHANG, J., AND GAO, J. Category-aware graph neural networks for improving e-commerce review helpfulness prediction. In *Proceedings of the 29th ACM International Conference on Information Knowledge Management* (New York, NY, USA, 2020), CIKM '20, Association for Computing Machinery, p. 2693–2700.
- [239] RACHERLA, P., AND FRISKE, W. Perceived ‘usefulness’ of online consumer reviews: An exploratory investigation across three services categories. *Electronic Commerce Research and Applications* 11, 6 (2012), 548–559.

- [240] RAMASWAMY, R., AND WOLF, T. High-speed prefix-preserving ip address anonymization for passive measurement systems. *IEEE/ACM Transactions on Networking* 15, 1 (2007), 26–39.
- [241] RAOOF, A., MATRAWY, A., AND LUNG, C.-H. Routing attacks and mitigation methods for rpl-based internet of things. *IEEE Communications Surveys & Tutorials* 21, 2 (2018), 1582–1606.
- [242] RATHGEB, C., BOTALJOV, A., STOCKHARDT, F., ISADSKIY, S., DEBIASI, L., UHL, A., AND BUSCH, C. Prnu-based detection of facial retouching. *IET Biometrics* 9, 4 (2020), 154–164.
- [243] RAWAT, D. B., DOKU, R., AND GARUBA, M. Cybersecurity in big data era: From securing big data to data-driven security. *IEEE Transactions on Services Computing* 14, 6 (2019), 2055–2072.
- [244] RAZA, S., DUQUENNOY, S., CHUNG, T., YAZAR, D., VOIGT, T., AND ROEDIG, U. Securing communication in 6lowpan with compressed ipsec. In *2011 International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS)* (2011), IEEE, pp. 1–8.
- [245] RAZA, S., WALLGREN, L., AND VOIGT, T. Svelte: Real-time intrusion detection in the internet of things. *Ad hoc networks* 11, 8 (2013), 2661–2674.
- [246] ŘEHŮŘEK, R., AND SOJKA, P. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks* (Valletta, Malta, May 2010), ELRA, pp. 45–50.
- [247] RING, M., WUNDERLICH, S., SCHEURING, D., LANDES, D., AND HOTHO, A. A survey of network-based intrusion detection data sets. *Computers & Security* 86 (2019), 147–167.
- [248] RÖSSLER, A., COZZOLINO, D., VERDOLIVA, L., RIESS, C., THIES, J., AND NIESSNER, M. FaceForensics++: Learning to detect manipulated facial images. In *International Conference on Computer Vision* (2019).
- [249] RUBINSTEIN, B. I., NELSON, B., HUANG, L., JOSEPH, A. D., LAU, S.-H., RAO, S., TAFT, N., AND TYGAR, J. D. Antidote: understanding and defending against poisoning of anomaly detectors. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement* (2009), pp. 1–14.
- [250] RUSSAKOVSKY, O., DENG, J., SU, H., KRAUSE, J., SATHEESH, S., MA, S., HUANG, Z., KARPATY, A., KHOSLA, A., BERNSTEIN, M., ET AL. Imagenet large scale visual recognition challenge. *International journal of computer vision* 115, 3 (2015), 211–252.
- [251] SAIF, H., HE, Y., AND ALANI, H. Semantic sentiment analysis of twitter. In *The Semantic Web - ISWC - 11th International Semantic Web Conference* (2012), pp. 508–524.
- [252] SALLAB, A. E., ABDOU, M., PEROT, E., AND YOGAMANI, S. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging* 13, 19 (2017), 70–76.
- [253] SAMANTA, S., AND MEHTA, S. Towards crafting text adversarial samples, 2017.

- [254] SANH, V., DEBUT, L., CHAUMOND, J., AND WOLF, T. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108* (2019).
- [255] SARHAN, M., LAYEGHY, S., MOUSTAFA, N., AND PORTMANN, M. Netflow datasets for machine learning-based network intrusion detection systems. In *Proc. EAI Int. Conf. Big Data Tech.* (2021), p. 117.
- [256] SCHAFFER, C. Selecting a classification method by cross-validation. *Machine Learning* 13, 1 (1993), 135–143.
- [257] SCHERER, K. R. What are emotions? and how can they be measured? *Social science information* 44, 4 (2005), 695–729.
- [258] SCHERHAG, U., DEBIASI, L., RATHGEB, C., BUSCH, C., AND UHL, A. Detection of face morphing attacks based on prnu analysis. *IEEE Transactions on Biometrics, Behavior, and Identity Science* 1, 4 (2019), 302–317.
- [259] SCHMIDT, A., AND WIEGAND, M. A survey on hate speech detection using natural language processing. In *Proceedings of the Fifth International Workshop on Natural Language Processing for Social Media* (2017), pp. 1–10.
- [260] SCHROEPFER, M. Creating a data set and a challenge for deepfakes. www.ai.facebook.com/blog/deepfake-detection-challenge/, 2019.
- [261] SEDJELMACI, H., SENOUCI, S. M., AND TALEB, T. An accurate security game for low-resource iot devices. *IEEE Transactions on Vehicular Technology* 66, 10 (2017), 9381–9393.
- [262] SHARAFALDIN, I., LASHKARI, A. H., AND GHORBANI, A. A. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *Proc. IEEE Int. Conf. Inf. Syst. Secur. Privacy* (2018), pp. 108–116.
- [263] SHARAFALDIN, I., LASHKARI, A. H., HAKAK, S., AND GHORBANI, A. A. Developing realistic distributed denial of service (ddos) attack dataset and taxonomy. In *2019 International Carnahan Conference on Security Technology (ICCST)* (2019), IEEE, pp. 1–8.
- [264] SHOKRI, R., STRONATI, M., SONG, C., AND SHMATIKOV, V. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)* (2017), IEEE, pp. 3–18.
- [265] SHU, K., SLIVA, A., WANG, S., TANG, J., AND LIU, H. Fake news detection on social media: A data mining perspective. *ACM SIGKDD explorations newsletter* 19, 1 (2017), 22–36.
- [266] SIDEY-GIBBONS, J. A., AND SIDEY-GIBBONS, C. J. Machine learning in medicine: a practical introduction. *BMC medical research methodology* 19, 1 (2019), 1–18.
- [267] SIMONYAN, K., AND ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
- [268] SINGH, V. P., AND PAL, P. Survey of different types of captcha. *International Journal of Computer Science and Information Technologies* 5, 2 (2014), 2242–2245.

- [269] SOMMER, R., AND PAXSON, V. Outside the closed world: On using machine learning for network intrusion detection. In *Proc. IEEE Symp. Secur. Privacy* (2010), pp. 305–316.
- [270] SONICWALL. Number of ransomware attacks per year 2022 | Statista — statista.com. <https://www.statista.com/statistics/494947/ransomware-attacks-per-year-worldwide/>. [Accessed 26-Sep-2022].
- [271] SPAGNOLETTIA, P., AND SALVIA, A. Digital systems in high-reliability organizations: Balancing mindfulness and mindlessness. In *Proc. Int. Workshop Socio-Techn. Perspective Inf. Syst. Develop.* (2020).
- [272] STATISTA. Media usage in an internet minute as of august 2020. <https://www.statista.com/study/12393/social-networks-statista-dossier> (2020). Accessed on 2021-03-18.
- [273] STATISTA. Total number of user reviews and opinions on tripadvisor worldwide from 2014 to 2021, feb 2022.
- [274] STEVANOVIC, M., AND PEDERSEN, J. M. An analysis of network traffic classification for botnet detection. In *Proc. IEEE Int. Conf. Cyber Situat. Awar., Data Analyt., Assessment* (Jun. 2015), pp. 1–8.
- [275] STOJKOSKA, B. L. R., AND TRIVODALIEV, K. V. A review of internet of things for smart home: Challenges and solutions. *Journal of cleaner production* 140 (2017), 1454–1464.
- [276] STOKES, J. K. The indecent internet: Resisting unwarranted internet exceptionalism combating revenge porn. *Berkeley Tech. LJ* 29 (2014), 929.
- [277] STONE, P. J., BALES, R. F., NAMENWIRTH, J. Z., AND OGILVIE, D. M. The general inquirer: A computer system for content analysis and retrieval based on the sentence as a unit of information. *Behavioral Science* 7, 4 (1962), 484.
- [278] SU, J., VARGAS, D. V., AND SAKURAI, K. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation* 23, 5 (2019), 828–841.
- [279] SU, Z., AND WASSERMANN, G. The essence of command injection attacks in web applications. In *Proceedings of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (2006), pp. 372–382.
- [280] SUN, N., ZHANG, J., RIMBA, P., GAO, S., ZHANG, L. Y., AND XIANG, Y. Data-driven cybersecurity incident prediction: A survey. *IEEE communications surveys & tutorials* 21, 2 (2018), 1744–1772.
- [281] SUN, S., LIU, Y., AND MAO, L. Multi-view learning for visual violence recognition with maximum entropy discrimination and deep features. *Information Fusion* 50 (2019), 43–53.
- [282] SURENDAR, M., AND UMAMAKESWARI, A. Indres: An intrusion detection and response system for internet of things with 6lowpan. In *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSP-NET)* (2016), IEEE, pp. 1903–1908.

- [283] SZEGEDY, C., IOFFE, S., VANHOUCHE, V., AND ALEMI, A. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2017), vol. 31.
- [284] TANG, J., GAO, H., HU, X., AND LIU, H. Context-aware review helpfulness rating prediction. In *Proceedings of the 7th ACM Conference on Recommender Systems* (New York, NY, USA, 2013), RecSys '13, Association for Computing Machinery, p. 1–8.
- [285] TARIQ, N., ASIM, M., MAAMAR, Z., FAROOQI, M. Z., FACI, N., AND BAKER, T. A mobile code-driven trust mechanism for detecting internal attacks in sensor node-powered iot. *Journal of Parallel and Distributed Computing* 134 (2019), 198–206.
- [286] TASPINAR, S., MOHANTY, M., AND MEMON, N. Source camera attribution using stabilized video. In *IEEE International Workshop on Information Forensics and Security* (2016), pp. 1–6.
- [287] TETCOS. NetSim by Tetcos. <https://www.tetcos.com/netsim-std.html>.
- [288] THELWALL, M., BUCKLEY, K., AND PALTOGLOU, G. Sentiment strength detection for the social web. *Journal of the American Society for Information Science and Technology* 63, 1 (2012), 163–173.
- [289] THOME, B., SHAMMA, D. A., FRIEDLAND, G., ELIZALDE, B., NI, K., POLAND, D., BORTH, D., AND LI, L.-J. Yfcc100m: The new data in multimedia research. *Communications of the ACM* 59, 2 (2016), 64–73.
- [290] THURASINGHAM, B. *Web data mining and applications in business intelligence and counter-terrorism*. CRC Press, 01 2003.
- [291] TIAN, Y., AND HOU, R. An improved aomdv routing protocol for internet of things. In *2010 International Conference on Computational Intelligence and Software Engineering* (2010), IEEE, pp. 1–4.
- [292] TOLOSANA, R., VERA-RODRIGUEZ, R., FIERREZ, J., MORALES, A., AND ORTEGA-GARCIA, J. Deepfakes and beyond: A survey of face manipulation and fake detection. *Information Fusion* 64 (2020), 131–148.
- [293] TRAMÈR, F., ZHANG, F., JUELS, A., REITER, M. K., AND RISTENPART, T. Stealing machine learning models via prediction {APIs}. In *25th USENIX security symposium (USENIX Security 16)* (2016), pp. 601–618.
- [294] TSUR, O., AND RAPPOPORT, A. Revrank: A fully unsupervised algorithm for selecting the most helpful book reviews. *Proceedings of the International AAAI Conference on Web and Social Media* 3, 1 (Mar. 2009), 154–161.
- [295] TSYMBAL, A. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin* 106, 2 (2004), 58.
- [296] VADICAMO, L., CARRARA, F., CIMINO, A., CRESCI, S., DELL'ORLETTA, F., FALCHI, F., AND TESCONI, M. Cross-media learning for image sentiment analysis in the wild. In *Proceedings of the IEEE international conference on computer vision workshops* (2017), pp. 308–317.

- [297] VELIOGLU, R., AND ROSE, J. Detecting hate speech in memes using multi-modal deep learning approaches: Prize-winning solution to hateful memes challenge. *arXiv preprint arXiv:2012.12975* (2020).
- [298] VERDOLIVA, L. Media forensics and deepfakes: An overview. *IEEE Journal of Selected Topics in Signal Processing* 14, 5 (2020), 910–932.
- [299] VINAYAKUMAR, R., ALAZAB, M., SOMAN, K., POORNACHANDRAN, P., AL-NEMRAT, A., AND VENKATRAMAN, S. Deep learning approach for intelligent intrusion detection system. *IEEE Access* 7 (2019), 41525–41550.
- [300] VINCENT, J. Watch jordan peele use ai to make barack obama deliver a psa about fake news. www.theverge.com/tldr/2018/4/17/17247334/ai-fake-news-video-barack-obama-jordan-peele-buzzfeed, 2018.
- [301] VINCENT, J. Facebook is now using ai to sort content for quicker moderation. www.theverge.com/2020/11/13/21562596/facebook-ai-moderation (2020). Accessed on 2020-11-13.
- [302] VON AHN, L., BLUM, M., HOPPER, N. J., AND LANGFORD, J. Captcha: Using hard ai problems for security. In *International conference on the theory and applications of cryptographic techniques* (2003), Springer, pp. 294–311.
- [303] VORMAYR, G., FABINI, J., AND ZSEBY, T. Why are my flows different? a tutorial on flow exporters. *IEEE Comm. Surv. Tut.* 22, 3 (2020), 2064–2103.
- [304] WALLGREN, L., RAZA, S., AND VOIGT, T. Routing attacks and countermeasures in the rpl-based internet of things. *International Journal of Distributed Sensor Networks* 9, 8 (2013).
- [305] WANG, G., WILSON, C., ZHAO, X., ZHU, Y., MOHANLAL, M., ZHENG, H., AND ZHAO, B. Y. Serf and turf: crowdturfing for fun and profit. In *Proceedings of the 21st international conference on World Wide Web* (New York, NY, USA, 2012), Association for Computing Machinery, pp. 679–688.
- [306] WASEEM, Z., AND HOVY, D. Hateful symbols or hateful people? predictive features for hate speech detection on twitter. In *Proceedings of the NAACL Student Research Workshop* (2016), pp. 88–93.
- [307] WILLIAMS, L., MCGRAW, G., AND MIGUES, S. Engineering security vulnerability prevention, detection, and response. *IEEE Software* 35, 5 (2018), 76–80.
- [308] WINTER, T., THUBERT, P., BRANDT, A., HUI, J. W., KELSEY, R., LEVIS, P., PISTER, K., STRUIK, R., VASSEUR, J.-P., ALEXANDER, R. K., ET AL. Rpl: Ipv6 routing protocol for low-power and lossy networks. *rfc 6550* (2012), 1–157.
- [309] WOLF, M., MILLER, K., AND GRODZINSKY, F. Why we should have seen that coming: Comments on microsoft’s tay “experiment,” and wider implications. *The ORBIT Journal* 1, 2 (2017), 1 – 12.
- [310] WOLF, T., DEBUT, L., SANH, V., CHAUMOND, J., DELANGUE, C., MOI, A., CISTAC, P., RAULT, T., LOUF, R., FUNTOWICZ, M., DAVISON, J., SHLEIFER, S., VON PLATEN, P., MA, C., JERNITE, Y., PLU, J., XU, C., LE SCAO, T., GUGGER, S., DRAME, M., LHOEST, Q., AND RUSH, A. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations* (Online, Oct. 2020), Association for Computational Linguistics, pp. 38–45.

- [311] WU, C.-J., BROOKS, D., CHEN, K., CHEN, D., CHOUDHURY, S., DUKHAN, M., HAZELWOOD, K., ISAAC, E., JIA, Y., JIA, B., ET AL. Machine learning at facebook: Understanding inference at the edge. In *Proc. IEEE Int. Symp. High Perf. Comp, Arch.* (2019), pp. 331–344.
- [312] WU, Y., SCHUSTER, M., CHEN, Z., LE, Q. V., NOROUZI, M., MACHEREY, W., KRIKUN, M., CAO, Y., GAO, Q., MACHEREY, K., KLINGNER, J., SHAH, A., JOHNSON, M., LIU, X., ŁUKASZ KAISER, GOUWS, S., KATO, Y., KUDO, T., KAZAWA, H., STEVENS, K., KURIAN, G., PATIL, N., WANG, W., YOUNG, C., SMITH, J., RIESA, J., RUDNICK, A., VINYALS, O., CORRADO, G., HUGHES, M., AND DEAN, J. Google’s neural machine translation system: Bridging the gap between human and machine translation, 2016.
- [313] XIAO, Q., CHEN, Y., SHEN, C., CHEN, Y., AND LI, K. Seeing is not believing: Camouflage attacks on image scaling algorithms. In *28th {USENIX} Security Symposium ({USENIX} Security 19)* (2019), pp. 443–460.
- [314] XIAO, Q., LI, K., ZHANG, D., AND XU, W. Security risks in deep learning implementations. In *2018 IEEE Security and privacy workshops (SPW)* (2018), IEEE, pp. 123–128.
- [315] XUAN, X., PENG, B., WANG, W., AND DONG, J. On the generalization of gan image forensics. In *Biometric Recognition* (2019), pp. 134–141.
- [316] YAN, J., AND EL AHMAD, A. S. A low-cost attack on a microsoft captcha. In *Proceedings of the 15th ACM conference on Computer and communications security* (2008), pp. 543–554.
- [317] YANG, Y., YAN, Y., QIU, M., AND BAO, F. Semantic analysis and helpfulness prediction of text for online product reviews. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)* (Beijing, China, 2015), Association for Computational Linguistics, pp. 38–44.
- [318] YAO, Y., VISWANATH, B., CRYAN, J., ZHENG, H., AND ZHAO, B. Y. Automated crowdturfing attacks and defenses in online review systems. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security* (New York, NY, USA, 2017), Association for Computing Machinery, pp. 1143–1158.
- [319] YAVUZ, F. Y., DEVRIM, Ü., AND ENSAR, G. Deep learning for detection of routing attacks in the internet of things. *International Journal of Computational Intelligence Systems* 12, 1 (2018), 39–58.
- [320] YE, G., TANG, Z., FANG, D., ZHU, Z., FENG, Y., XU, P., CHEN, X., AND WANG, Z. Yet another text captcha solver: A generative adversarial network based approach. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (2018), pp. 332–348.
- [321] YOU, Y., ZHANG, Z., HSIEH, C.-J., DEMMEL, J., AND KEUTZER, K. ImageNet training in minutes. In *Proc. Int. Conf. Parallel Proces.* (2018), pp. 1–10.
- [322] YUAN, K., TANG, D., LIAO, X., WANG, X., FENG, X., CHEN, Y., SUN, M., LU, H., AND ZHANG, K. Stealthy porn: Understanding real-world adversarial images for illicit online promotion. In *2019 IEEE Symposium on Security and Privacy (SP)* (2019), pp. 952–966.

- [323] ZANELLA, A., BUI, N., CASTELLANI, A., VANGELISTA, L., AND ZORZI, M. Internet of things for smart cities. *IEEE Internet of Things journal* 1, 1 (2014), 22–32.
- [324] ZHANG, C., COSTA-PÉREZ, X., AND PATRAS, P. Tiki-taka: Attacking and defending deep learning-based intrusion detection systems. In *Proc. ACM Conf. Cloud Com. Secur. Workshop* (2020), pp. 27–39.
- [325] ZHANG, W. E., SHENG, Q. Z., ALHAZMI, A., AND LI, C. Adversarial attacks on deep-learning models in natural language processing: A survey. *ACM Transactions on Intelligent Systems and Technology (TIST)* 11, 3 (2020), 1–41.
- [326] ZHANG, X., AND GHORBANI, A. A. An overview of online fake news: Characterization, detection, and discussion. *Information Processing & Management* 57, 2 (2020), 102025.
- [327] ZHANG, Y., NIU, J., HE, G., ZHU, L., AND GUO, D. Network Intrusion Detection Based on Active Semi-supervised Learning. In *Proc. IEEE/IFIP Int. Conf. Dep. Syst. Netw. Workshops* (2021), pp. 129–135.
- [328] ZHANG, Y., RUAN, X., WANG, H., WANG, H., AND HE, S. Twitter trends manipulation: a first look inside the security of twitter trending. *IEEE Transactions on Information Forensics and Security* 12, 1 (2016), 144–156.
- [329] ZHANG, Z., AND VARADARAJAN, B. Utility scoring of product reviews. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management (New York, NY, USA, 2006), CIKM '06, Association for Computing Machinery*, p. 51–57.
- [330] ZHAO, Y., NASRULLAH, Z., AND LI, Z. Pyod: A python toolbox for scalable outlier detection. *Journal of Machine Learning Research* 20, 96 (2019), 1–7.
- [331] ZHOU, S., AND GUO, B. The order effect on online review helpfulness: A social influence perspective. *Decision Support Systems* 93 (2017), 77–87.