Ca' Foscari
University
of Venice

Ph.D. Degree
in Computer Science

Cycle XXXV

Final Thesis

Vulnerability of Machine Learning: A Study on Poisoning Attacks

Supervisor
Prof. Marcello Pelillo

Co-supervisor
Prof. Battista Biggio

Graduand
Antonio Emanuele Cinà
Matriculation Number 854866

Academic Year 2022 / 2023

Abstract

Training cutting-edge machine learning models is often prohibitive to most because it requires expensive hardware and a huge amount of labeled data. To address this shortcoming, pre-trained models or publicly-available data are employed to reduce the financial development costs. However, these practices are becoming the Achilles' heel of the machine learning development chain because they expose the models to poisoning. These attacks assume the capacity of the attacker to tamper with the model training or data collection phases to drive the model toward unexpected misclassifications at test time. Notably, poisoning attacks are the most feared threat by companies because of the harm they can cause and the difficulties in detecting them. Therefore, mindful monitoring of the data gathering and model training procedures is becoming imperative. Due to the practical relevance of poisoning attacks, various scientific articles have been published about this topic. However, despite this considerable interest, we found a lot of confusion, misconceptions, and open questions that we investigate in this thesis. We thus tackle five different research questions, namely: (1) how to categorize poisoning attacks; (2) how to make them scalable in practice (3) how can we analyze them and understand the factor influencing their effectiveness against ML models; (4) how poisoning can influence other ML aspects, going beyond misclassification violations; and (5) how an attacker can craft poisoning samples when having access to the target system only by querying it. For each of these research questions, we revisit the underlying problem, the state of the art going in that research direction, and we present the contributions proposed by the author of this thesis for answering them. Finally, we shed light on the current limitations and open research questions in this field and propose possible future research directions. I believe that the findings in this thesis will help the ML community to better evaluate the risks of poisoning attacks and stimulate the discussion towards developing novel benchmarks and defensive techniques to protect ML models.

Contents

Symbols **v**

1 Introduction **1**
- 1.1 Research Questions 3
- 1.2 Thesis Outline and Contributions 5
- 1.3 List of Publications 5

2 Background Concepts **7**
- 2.1 Machine Learning 7
  - 2.1.1 Supervised Learning 7
  - 2.1.2 Unsupervised Learning 12
- 2.2 Adversarial Machine Learning 15
  - 2.2.1 Exploratory Attacks 16
  - 2.2.2 Causative Attacks 17

3 Machine Learning Security against Data Poisoning **19**
- 3.1 Poisoning in Real-World Applications 20
- 3.2 Modeling Poisoning Attacks and Defenses 21
  - 3.2.1 Learning Settings 22
  - 3.2.2 Attack Framework 22
  - 3.2.3 Defense Framework 25
  - 3.2.4 Poisoning Attacks and Defenses 26
- 3.3 Poisoning Attacks 26
  - 3.3.1 Indiscriminate (Availability) Poisoning 27
  - 3.3.2 Targeted (Integrity) Poisoning 30
  - 3.3.3 Backdoor (Integrity) Poisoning 32
  - 3.3.4 Unifying Framework 34
  - 3.3.5 Development Timeline 34
  - 3.3.6 Transferability of Poisoning Attacks 35
- 3.4 Defenses Against Poisoning 36
  - 3.4.1 Matching Poisoning Attacks and Defenses 37
- 3.5 Poisoning in Other Domains 38
- 3.6 Concluding Remarks 39

4 Improving Scalability of Data Poisoning **41**
- 4.1 BetaPoisoning 42
- 4.2 Experimental Analysis 44
- 4.3 Concluding Remarks 49

5 Understanding Backdoor Poisoning Vulnerability 51
5.1 Backdoor Learning Process . . . 52
5.2 Experimental Analysis . . . 55
5.3 Concluding Remarks . . . 67

6 Causing Energy-Latency Failures via Poisoning 69
6.1 Sponge Poisoning . . . 70
6.2 Experimental Analysis . . . 75
6.3 Concluding Remarks . . . 82

7 Poisoning Clustering Under Limited Knowledge 83
7.1 Poisoning Clustering via Adversarial Queries . . . 84
7.2 Experimental Analysis . . . 89
7.3 Concluding Remarks . . . 96

8 Open Challenges and Conclusions 99
8.1 Current Limitations . . . 99
8.2 Future Development . . . 100
8.3 Conclusions . . . 101

Bibliography 105

A Understanding Backdoor Poisoning Vulnerability 125

B Causing Energy-Latency Failures via Poisoning 141

C Poisoning Under Limited Knowledge 145

Symbols

Data notation

$\mathcal{D}$	training pristine data
$\mathcal{D}_p$	training poisoning data
$\mathcal{D}'$	training dataset ($\mathcal{D}' = \mathcal{D}_p \cup \mathcal{D}$)
$\mathcal{V}$	Validation/Surrogate pristine data
$\mathcal{V}_t$	Validation/Surrogate poisoning data
$\mathcal{V}'$	Validation/Surrogate data ($\mathcal{V}' = \mathcal{V}_t \cup \mathcal{V}$)
$\mathcal{T}$	Test data
$\mathcal{T}_p$	Test data containing backdoor trigger
$\boldsymbol{x}$	Input sample $\boldsymbol{x} \in \mathbb{R}^d$
$\mathbf{y}$	Ground true label $y \in \{0, \dots, C-1\}$
$\hat{y}$	Predicted label $\hat{y} \in \{0, \dots, C-1\}$
$\boldsymbol{x}_{\mathrm{lb}}$	Lower bound of the feature space $\boldsymbol{x}_{\mathrm{lb}} \in \mathbb{R}^d$
$\boldsymbol{x}_{\mathrm{ub}}$	Upper bound of the feature space $\boldsymbol{x}_{\mathrm{ub}} \in \mathbb{R}^d$
$\boldsymbol{X}$	Feature matrix of data samples $\boldsymbol{X} \in \mathbb{R}^{n \times d}$

Model and training notation

f	Model's classification layers
ϕ	Model's feature extractor
$\mathcal{M}$	Machine learning model ($\mathcal{M} = f \circ \phi$)
$\boldsymbol{\theta}$	Model's parameters
$\boldsymbol{\theta}^\star$	Optimal model's parameters
L	Loss function
$\mathcal{L}$	Regularized loss function
$\nabla_{\boldsymbol{x}} L$	Gradient of loss function L w.r.t. $\boldsymbol{x}$
$\mathcal{C}$	Generic clustering algorithm

Ringraziamenti

Immaginate di essere al Luna Park in attesa di provare le montagne russe che tanto attendavate (nel mio caso le Raptor a Gardaland). La gioia nel vederti li, a pochi passi dalla giostra, cresce sempre di più! Allo stesso tempo tuttavia cresce anche la tua insicurezza, o paura in essa. Ce la farò a resistere? Ne sono in grado? Poi sali, il giro inizia, VIA. Un po' di discese si, ma anche un po' di salite. Fine del giro, sei ancora più felice di prima e vorresti riprovare il giro. Adesso, perché questa storia in una tesi di dottorato? Perché il mio percorso di dottorato voglio ricordarlo esattamente cosi. Ricco di gioia già prima che iniziasse, ma con quel tocco di ansia extra che ti porta a grosse riflessioni su te stesso e il tuo operato. E' stata un'esperienza grandiosa, dove ho avuto l'occasione di conoscere molto persone, ma sopratutto LA REALTÀ che fa per me. Le discese hanno sicuramente fatto male, ma le salite mi hanno dato un'infinita carica che porterò sempre con me. Sapete un'altra analogia tra la montagna russa e il dottorato? Entrambe le giostre le affronti con qualcuno! E per questo voglio ringraziarli tutti di seguito.

Prima di tutto vorrei ringraziare il mio supervisore Marcello Pelillo e il co-supervisore Battista Biggio. Al primo devo il merito di avermi fatto innamorare della figura del Professore. Da suo studente ho sempre amato le sue lezioni, difficili assolutamente, ma affrontate nel migliore dei modi. Eleganza, consapevolezza, ma sopratutto PASSIONE. Il miglior augurio che posso farmi è di riuscire ad ereditare questa sua capacità, e di poterla a mia volta trasmettere. A Battista e al suo team di ricerca devo grand parte della mia conoscenza in ambito adversarial. Mi ha accolto come un fringuello alle prime armi, e mi hanno instradato verso un percorso di formazione più chiaro. Battista in primis ha insegnato a rivalutare le mie idee, a rimetterle in discussione, a ri-organizzarle mentalmente, come comunicarle meglio, ma sopratutto mi ha insegnato ad avere senso critico e costruttivo nei confronti della ricerca.

E poi abbiamo Ambra Demontis. Ecco, qui potrei iniziare e mai finire. A lei devo tantissimo! Mi è sempre stata vicina, e se non sono finito in reparto di psichiatria lo devo solo a lei :-). Lei mi ha instradato verso la strada del *self-improvement* e della gestione dello stress, che mi stava veramente divorando. Oggi sono molto più felice, tranquillo, e meno ansioso. Ho ancora molta strada da fare, ma qui bisogna fare i complimenti ad Ambra per essere riuscita a recuperare un caso speciale come me.

Voglio ringraziare anche i miei colleghi e amici a Venezia, purtropo la lista è lunga ma voglio citare sicuramente Alessandro e Sebastiano. Lo so ragazzi, vi ho portato al limite. Ma grazie, mi avete insegnato molto e per questo ve ne sarò grato sempre.

Poi abbiamo gli "Special Ones", Maura e Luca. Si voi. Sapete che sto parlando di voi. Tra fioricini, scoiattoli, e oggetti poco CHIARI ho sempre trovato il sorriso con voi e questo non potrò mai dimenticarlo. Sono stato davvero bene con voi a Cagliari, ed è sempre un piacere ritrovarvi. Siete magici! A Luca voglio solo dire "Preparà il mirto, abbiamo una serata da replicare.". A Maura invece "Prenota il tavolo da Su' Cumbido, qualche piatto sta aspettando solo noi".

Sempre da Cagliari (passando per la Germania in realtà) con furore abbiamo Kathrin Grosse. Con lei ho condiviso moltissimo del mio percorso di ricerca. È stata una bella sfida scrivere la survey insieme ma alla fine siamo veramente orgogliosi del lavoro fatto e siamo convinti lo saranno anche i nostri lettori. Grazie Kathrin, sono cresciuto moltissimo anche grazie a te!
Siamo agli sgoccioli, ma c'è una persona che voglio ringraziare tantissimo. Non ci siamo mai visti di persona (purtroppo) ma è stata molto presente in questi anni di collaborazione, e questa persona è Fabio Roli. Fabio è per me l'emblema di come una persona possa migliorare se stessa fino a raggiungere traguardi altissimi. Gli insegnamenti di Fabio insieme a quelli di Ambra riguardo allo sviluppo personale li porterò sempre avanti. Grazie davvero per avermi reso una persona migliore. O almeno meno ansiosa.

Voglio ringraziare Giacomo e David con cui sto portando avanti un bel progettino, e con cui sto condividendo esperienze pazzesche. Ragà mi raccomando! Sto HackZurich 2023 dobbiamo portarlo a casa, cosi come il tornero di ping-pong. Alleniamoci, ma occhio a Framly ;-)

Dulcis in fundo, voglio ringraziare la mia compagna Silvia, i miei genitori, mio fratello, e mia sorella. Per Silvia non ci sono veramente parole che potranno mai essere sufficienti per descrivere quanto lei mi sia stata vicina in questi anni, ma sopratutto il suo grandissimo sostegno. È scontato dirlo, ma grazie. So che preferisci un sushi di ringraziamento piuttosto che le parole scritte, ma intanto iniziamo da qua.Passando alla mia famiglia, purtroppo sono stato poco presente negli ultimi anni, e questo loro non lo meritano. Mi dispiace, ma sappiate che vi voglio molto bene e farò in modo di recuperare questo tempo perso.Per i restanti (e ce ne sarebbero...) non me ne vogliate, è tarda notte e potrei scrivere altre 20 pagine per ringraziarvi tutti. Non so però poi quanto sarebbero felici i miei revisori di leggere tutto questo. Molte persone meritano il mio riconoscimento, ma questo può andare anche al di fuori di queste poche righe.

Chapter 1

Introduction

The state of the art in machine learning (ML) has considerably improved over the last decade. Learning models are nowadays widely employed as practical aiding tools for many data services. Their adoption is becoming the de facto standard for diverse tasks, such as natural language processing [297], face recognition [265], cancer detection [187], and road sign recognition [262], as vital tools for data analysis and autonomic decision-making. They are also used in many cybersecurity applications, such as spam email filtering [186; 193] or malware detection [217].

The success of machine learning in these applications has been achieved both in supervised contexts, where classification and regression models prevail, and in unsupervised contexts, with clustering and data dimensionality reduction algorithms. The key strength of learning models is their ability to infer patterns that can be used for future predictions, especially when there is a massive amount of unknown test data to process. This makes them particularly well-suited to domains where the implementation is too complex to be designed manually but for which a large amount of data are given. An incentive for the advancement of this technology has been given by the increasing availability of specialized computing resources (e.g., GPUs) and cloud computing services that make ML more accessible and enable ever higher performance.

Adversarial Machine Learning. Nevertheless, ML has traditionally been developed under the assumption that the environment is benign during training and usage of the model. Most of the learning algorithms assume that the data used for training them are representative of the data encountered at test time. Moreover, they further assume that the noise inside data is randomly distributed. These assumptions have been helpful for designing efficacious ML models, but they do not cover the case where malicious users try to alter this condition to reach their goal. Therefore, the increasing pervasiveness of ML algorithms in high-stake real and critical applications poses an issue about their robustness in the presence of malicious manipulations. For example, in 2005, Lowd and Meek [180] showed that malicious users could evade ML-based spam email filters by inserting or appending words indicative of legitimate email. Since then, *adversarial learning* has emerged as a line of research focused on studying how machine learning can be broken and how to prevent that from happening [17; 23; 62; 63; 179]. From that time, new security threats have been discovered in the past years that can compromise *availability*, *integrity*, and *confidentiality* of ML systems [23].

Attacks on *availability* attempt to decrease the performance (e.g., prediction accuracy or latency) victim system or neglect its access to legitimate users (e.g., denial of service). For example, an attacker may tamper with the ML spam filter of the victim to decrease its accuracy, thus making any spam emails pass the filter more easily. Attacks on *integrity* induce the model behavior towards a specific direction chosen by the attacker. For example, the attacker manipulates the content of an email so that it will pass the spam filtering defense. Finally, *confidentiality* attacks aim to steal information about the training dataset or the deployed ML model. For example, the attacker may try to retrieve sensible information from the training dataset (e.g., data about hospital patients) or reconstruct the ML decision model used by a competitor company.

Attacks against ML. These violations against ML can be caused by malicious users through different attacks, and the most investigated are *evasion* and *poisoning.*

Evasion attacks aim to maliciously perturb the input data to have them misclassified by the model, causing an integrity violation as for the spam example seen above. The resulting samples are called adversarial examples, and their existence questions the robustness and reliability of ML systems. Researchers have shown that even invisible perturbations - unrecognizable to human eyes - can cause misclassification in state-of-the-art models [28; 38; 105].

Poisoning attacks are well-known security threats caused by adversaries who are enabled to inject malicious samples in the training data or tamper with the training algorithm to change the expected behavior of the victim's model.

Conversely to evasion attacks, poisoning attacks have been investigated in all three violation directions. *Indiscriminate* poisoning attacks aim to cause availability violations that, in most cases, translate into a significant accuracy reduction in the victim's model for any test time sample. *Targeted* and *backdoor* poisoning, instead, leads the model to misclassify some test samples, causing an integrity violation. The distinction between target and backdoor is that the former brings the model to misclassify only a few specific target samples, while the latter brings the model to misclassify all test samples containing a backdoor trigger known only to the attacker. Notably, the state-of-the-art attacks are implemented following many different strategies, some computationally scalable[1] and some not, and under a multitude of assumptions for the attacker. However, the huge plethora of implementations and the lack of clear systematization has often led to confusion in the literature, inducing researchers to attacks developed with quite different assumptions and thus making unfair comparisons, [63; 236]. Furthermore, three aspects in the poisoning literature remain unclear, i.e., (i) which are factors affecting the poisoning efficacy, (ii) whether poisoning attacks can go beyond misclassification, and (iii) if poisoning can be staged against unsupervised clustering algorithms under more realistic assumptions. Regarding the first aspect, it is unclear why poisoning attacks are often effective in the scenarios in which they are thought of. The factors influencing the effectiveness of a poisoning attack, or the vulnerability of a model to this threat, are still unclear. Eventually, this knowledge could lead to significant developments in improving attack performance or developing new defenses.

The second aspect arose as a result of recent developments in machine learning security. In 2020 a new attack vector against ML was identified, namely *sponge examples.* The main novelty of the Shumailov et al. [245]'s work is that it goes beyond degrading the accuracy performance of the learning models. Unlike evasion and poisoning attacks, sponge attacks do not compromise the model's accuracy but rather its usability or the lifetime of the battery on which it is run. Sponge examples are test-time inputs carefully-optimized to increase energy consumption and latency of DNNs when deployed on hardware accelerators. Consequently, processing these malicious samples will drain the system's batteries faster, increase the prediction latency, and decrease the throughput, compromising its availability to legitimate users. For example, an attacker can forward multiple sponge examples to the target system to cause a denial of service by making the target unresponsive due to the ever-increasing latency in the decisions. However, we are still in the early days of this new research front; the attack developed so far is carried out only in the testing phase, assuming the capacity of the attacker to craft a series of sponge examples that will then be forwarded to cause the energy-latency violation. Query-tracking defensive techniques [49; 138], which monitor user queries and block illicit activities, could be adopted to limit their application and effectiveness. A possible solution, yet to be explored in the literature, is to stage the attack before the model is deployed through poisoning attacks. It is not known what the developments of this new frontier may be, but it is undoubtedly a new interesting threat model that no longer see the ML model as the primary subject of attack but also the system in which it is employed.

Finally, the third aspect stems from a little exploration of poisoning attacks in the unsupervised

---

[1] An attack is scalable if it can handle, from an execution time point of view, the increasing complexity of the target model.

learning context. Although interest in machine learning security is growing, most work has focused on examining the robustness of supervised learning models. The unsupervised counterpart is relatively little explored. Only a few papers have contributed to this research direction, highlighting the technical difficulties in proposing novel attacks and defenses. Because of this difficulty, however, the underlying assumptions introduced to facilitate the problem are so strong to limit their application. For example, most papers attacking clustering algorithms rely on the assumption that the attacker perfectly knows the data to corrupt and the target clustering algorithm. Knowing these two components allows the attacker to stage the attack taking advantage of the known vulnerability and behavior of the target algorithm. Conversely, attacks with more limited knowledge represent a more challenging and practical scenario, however, this research direction is still under-researched.

Due to the relevance and potential risks caused by evasion, poisoning, or sponge attacks, securing ML services against them is necessary during every step of an ML life cycle. From data collection to deployment, ML models have been discovered to be fragile and such vulnerabilities must be repaired, as in traditional software development [150]. However, there is still a long way to go in securing learning models, partially due to the lack of attacks for deeply testing their vulnerabilities. Some open challenges still limit the progress of this research direction. Therefore, there is a need to recognize these challenges and suggest possible research directions to foster the development of novel tools for identifying ML vulnerabilities.

1.1 Research Questions

To promote the above vision, this thesis aims to shed light on the types of poisoning attacks that can be staged, the factors influencing their effectiveness, their scalability, and open challenges that are limiting their development. We further explore a novel poisoning threat against ML, extending the concept of sponge examples to the poisoning case. We present our work on poisoning attacks by answering the five research questions (RQ) presented below. For each of them, we discuss its relevance, explore the corresponding related work, and present our contributions to answer the question.

Research Question #1

How can we categorize/distinguish poisoning attacks against ML?

We argue that the literature around poisoning is often quite chaotic. The distinction between the different poisoning attack types is unclear, leading to unfair comparisons in the experimental evaluation. In some works, for example, we can see comparisons between attacks with essentially different goals and assumptions that require different evaluations. Moreover, proper categorization of when poisoning attacks can be staged and under which attackers' capabilities and knowledge are missing. Therefore, we questioned on the feasibility of developing a good framework that enables a wise categorization of poisoning attacks under their different attack strategies and assumptions.

Research Question #2

Can we make poisoning attacks scalable?

One of the critical aspects of systems vulnerabilities analysis is to have a complete evaluation framework, i.e., a benchmark, to test the robustness and reliability of ML models in the presence of malicious adversaries. From a poisoning perspective, a benchmark should execute a series of attacks and measure the model's robustness against them. However, this procedure requires that attacks to be feasible and executable under various conditions, such as increasing of the model's complexity.

This thesis explores the scalability property of poisoning attacks under different attacker capabilities and sheds light on where existing approaches are limited and how researchers in the future may improve them.

Research Question #3

Which factors influence the effectiveness of backdoor poisoning attacks?

Understanding the factors influencing the vulnerability of a ML model against poisoning attack may represent the first step towards developing novel defensive techniques to protect them. We, however, realized that at the state of the art most works in this direction evaluate only availability and targeted attacks, and little work has been done for backdoors. With this question, we want to stimulate the discussion on the properties of ML models that make them vulnerable to this threat. Finding the factors influencing their vulnerability may be crucial for developing novel defensive techniques for protecting ML against backdoor threats.

Research Question #4

Can we increase energy-latency performance of ML models via poisoning?

Most of the literature around poisoning attacks is limited in trying to compromise the prediction accuracy of the victim model. Whether other kinds of violations can be inflicted on the victim model through poisoning are unknown or little explored. Inspired by the novelty introduced by Shumailov et al. [245] in designing the sponge example attack, this thesis explores a new frontier for poisoning attacks aimed at increasing energy consumption in the victim model.

Research Question #5

Can we poison clustering algorithms under limited knowledge?

Testing the robustness of clustering algorithms has been proved to be a complex task [29; 31]. Existing works were able to demonstrate the fragility of these algorithms by assuming high knowledge on the attacker about the clustering algorithm to corrupt [29; 31; 55; 68]. However, little or no works tried to evaluate the feasibility of crafting poisoning samples against these algorithms, assuming the attacker has limited knowledge about the target algorithm. Such an analysis can offer a worst-case measure of robustness against malicious users, and it can also serve to find common weaknesses between clustering algorithms. Therefore, we explore if and how an attacker can exploit oracle-access[2] to the target algorithm for devising a poisoning attack.

Research Question #6

Which are the open challenges in the poisoning ML literature?

Finally, in this thesis we also investigate the still open challenges in the poisoning literature that, if solved, could positively contribute to the realization of robust defensive techniques, guidelines for secure data gathering, or benchmarks to test the robustness against such attacks. We further

---

[2] The attacker can only query the system as a service.

investigate on which could be the possible research directions from which researchers can take inspiration to solve them.

1.2 Thesis Outline and Contributions

Chapter 2 gives some background and context information about machine learning and adversarial machine learning.

In Chapter 3, we respond to research question #3 by offering an overview of the state of the art of poisoning attacks and proposing a framework to categorize existing attacks and defenses based on the attacker and defense threat models. This analysis results from our paper **Wild Patterns Reloaded: A Survey of Machine Learning Security against Training Data Poisoning.** This paper has been written with the equal contribution of my colleague and friend Kathrin Grosse. The author of this thesis was responsible for developing the threat models for poisoning attacks and their categorization, while Kathrin worked mainly on the part of defenses against poisoning. Finally, we collaborated to match the defenses with the attacks they might prevent.

Additionally, in Chapter 3 we revisit the paper **Machine Learning Security against Data Poisoning: Are We There Yet?** where we study the implications of these attacks against the Trustworthiness of AI models, an essential requirement planned by the European Union for the future production of systems employing AI models [89].

The remaining of this thesis is organized following the research questions presented before. From Chapter 4 to Chapter 6 we reveal our contributions in the directions defined by the research questions (RQ) #2,#3, and #4. Specifically, in Chapter 4 we argue on the scalability issues of existing poisoning attacks (related to RQ #2) presented in our paper **The Hammer and the Nut: Is Bilevel Optimization Really Needed to Poison Linear Classifiers?** and present a heuristic attack that may be employed to reduce the natural complexity of the problem when certain assumptions are met.

We then present in Chapter 5 our framework developed in **Backdoor Learning Curves: Explaining Backdoor Poisoning Beyond Influence Functions** for analyzing the factors influencing the vulnerability against poisoning attack, thus responding to RQ #3.

Finally, in Chapter 6 we respond to RQ #4 by presenting our sponge poisoning attack, designed in the paper **Energy-Latency Attacks via Sponge Poisoning**, which goes beyond the standard misclassification violation of existing poisoning attacks and causes an energy-latency violation.

Our contribution to poisoning in unsupervised domain in reported in Chapter 7. We present our paper **A black-box adversarial attack for poisoning clustering** where we test the robustness of unsupervised clustering algorithms in the presence of malicious users. Moreover, we respond to our RQ #5 by showing that evaluating the robustness of clustering algorithms can be complex and becomes even more complicated if the attacker's knowledge about the target system is reduced.

Finally, in Chapter 8, we present the main open challenges in the poisoning attacks literature that limit their development. We further suggest possible future directions that may represent possible opportunities to solve them, and conclude with a summary of the main achievements and limitations of this thesis.

In Table 1.1, we summarize the relationship between the research questions we pose in this thesis, the thesis's chapters, and the author's publications.

1.3 List of Publications

This thesis is based on the following publications:

- **Cinà, Antonio Emanuele**, Ambra Demontis, Battista Biggio, Fabio Roli and Marcello Pelillo. "Energy-Latency Attacks via Sponge Poisoning." *Submitted to Transaction on Pattern Analysis and Machine Intelligence (TPAMI)* (2022).

	Chapter	RQ#1	RQ#2	RQ#3	RQ#4	RQ#5	RQ#6
Publication II [63]	3, 8	✔					✔
Publication II [62]	3, 8	✔					✔
Publication III [60]	4		✔				
Publication IV [59]	5			✔			
Publication V [61]	6				✔		
Publication VI [64]	7					✔	

Table 1.1: *Relation between research questions, chapters, and publications.*

- **Cinà, Antonio Emanuele**, Kathrin Grosse, Ambra Demontis, Battista Biggio, Fabio Roli and Marcello Pelillo. "Machine Learning Security against Data Poisoning: Are We There Yet?" *Submitted to IEEE Computer* (2022).
- **Cinà, Antonio Emanuele**, Kathrin Grosse, Ambra Demontis, Sebastiano Vascon, Werner Zellinger, Bernhard Alois Moser, Alina Oprea, Battista Biggio, Marcello Pelillo and Fabio Roli. "Wild Patterns Reloaded: A Survey of Machine Learning Security against Training Data Poisoning." *Submitted to ACM Computing Survey* (2022).
- **Cinà, Antonio Emanuele**, Kathrin Grosse, Sebastiano Vascon, Ambra Demontis, Battista Biggio, Fabio Roli and Marcello Pelillo. "Backdoor Learning Curves: Explaining Backdoor Poisoning Beyond Influence Functions." *Submitted to Computers & Security* (2022).
- **Cinà, Antonio Emanuele**, Sebastiano Vascon, Ambra Demontis, Battista Biggio, Fabio Roli and Marcello Pelillo. "The Hammer and the Nut: Is Bilevel Optimization Really Needed to Poison Linear Classifiers?" *International Joint Conference on Neural Networks (IJCNN)* (2021).
- **Cinà, Antonio Emanuele**, Alessandro Torcinovich and Marcello Pelillo. "A Black-box Adversarial Attack for Poisoning Clustering." *Pattern Recognition* (2022).

Chapter 2

Background Concepts

The state of the art of machine learning models has gained an incredible ability to make good predictions in different domains, like image classification, speech recognition, market analysis, intrusion detection, and malware detection. Due to their outstanding results, learning systems carry fundamental roles in sophisticated applications as tools for aiding decision-making. However, it has been observed that, despite their very high learning capacity, they are sensible to make wrong predictions when data are maliciously perturbed. A surprising observation is that sometimes these alterations are invisible to human eyes, questioning how these models handle data. The critical assumption that causes such vulnerability is that these models have not been designed for working in scenarios where an attacker wants to subvert or compromise the results of the system. They are trained in safe environments, assuming data noise is randomly distributed and no users intentionally pollute them.

In this chapter, we will provide background concepts on machine learning and adversarial machine learning, and we will introduce the notation that will be used for the rest of the thesis.

2.1 Machine Learning

Machine learning (ML) studies algorithms and mathematical models that learn to perform a particular task through data and experience. The key finding is that ML models are not explicitly programmed to solve the task but retrieve patterns in the data that are then employed for making predictions. The more data they are fed, the more accurate or complete their response is. ML methods can be classed as supervised, unsupervised, and reinforcement learning. In this thesis, we will focus on *classification* and *clustering* problems, a family of algorithms that falls respectively in the supervised and unsupervised learning categories.

2.1.1 Supervised Learning

In supervised learning, ML modes pass through three phases: training, validation, and testing. During training, a labeled dataset $\mathcal{D}'$ is used to estimate a mapping function $\mathcal{M}$ that solves the target task. The training dataset $\mathcal{D}'$ consists of input-output pairs $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^{i=n}$, where $\boldsymbol{x}_i$ is a single input sample, and y_i represents the corresponding output. At validation time, the model's performance (e.g., prediction accuracy) is estimated on novel validation dataset $\mathcal{V}'$. Validation is necessary to assess that the model can generalize its prediction ability also on data seen during training. Finally, if the validation phase has given acceptable results, the model is deployed and used for predicting novel unknown test data $\mathcal{T}$. The fundamental assumption for supervised learning models is that training, validation, and test data are sampled from the same data and labels distributions $\mathcal{X}$and $\mathcal{Y}$.

A *classifier* is a supervised machine learning algorithm that, given an input sample, outputs one class among the C prefixed by the problem. Mathematically speaking, it learns a mapping $\mathcal{M}: \boldsymbol{x}, \boldsymbol{\theta} \mapsto y$, where $\boldsymbol{x} \in \mathbb{R}^d$ is a feature vector containing d measurable attributes describing the object (e.g., pixel intensity, number of transactions, etc.), $\boldsymbol{\theta}$ the model's parameters, and y is the classification output. For each of the C classes in the learning task, the model output confidence scores, and the final classification is achieved by taking:

$$\hat{y} = \underset{C \in \{0,\ldots,C-1\}}{\operatorname{argmax}} \mathcal{M}(\boldsymbol{x}; \boldsymbol{\theta})$$

as the predicted label.

Training, Validation, and Test. The learning (or training) phase consists in determining a configuration of weights $\boldsymbol{\theta}$ such that the model output should be as close as possible to the desired output as many times as possible for all the examples in the training set. In other words, the training phase aims at creating a model that has learned patterns in the training data for correctly classifying most of the time. Typically, this amounts to minimizing an **loss function** $\ell : (\boldsymbol{x}, y; \boldsymbol{\theta}) \mapsto \mathbb{R}$, that measures the cost of classifying instance $\boldsymbol{x}$ as y over the entire training set. A *loss function* is a function that returns a numerical value typically proportional to the difference between the desired outputs and the output produced by the model on a set of samples. Classifiers can adopt different losses, with some penalizing certain behaviors and some not, but they all have in common the idea of assigning high scores when the model's prediction on $\boldsymbol{x}$ is different from the true label y. The training algorithm thus aims to find the optimal model's parameters $\boldsymbol{\theta}^\star$ which minimize the risk for the classifier to have misclassifications on the training dataset.

$$\boldsymbol{\theta}^\star \in \arg\min \frac{1}{n} \sum_{\{\boldsymbol{x}_i, y_i\} \in \mathcal{D}'} \ell(\boldsymbol{x}_i, y_i, \theta)$$

This process is also named **empirical risk minimization**.

Once the model has been trained, the subsequent steps are validation and testing. During validation, we evaluate the model's accuracy, i.e., the percentage of correctly classified samples. Given the validation dataset $\mathcal{V}'$, the validation accuracy for model $\mathcal{M}$ is mathematically formulated as:

$$\mathcal{A}(\mathcal{M}, \mathcal{V}') = \frac{1}{N} \sum_{i=1}^{n} \mathbb{1}_{\hat{y}_i = y_i}$$

where $\hat{y}_i$ is the label predicted by model $\mathcal{M}$ for input $\boldsymbol{x}_i$ in the validation dataset.
Validation accuracy can be used to test the model's performance on data distinct from the one used during training but sampled from the same underlying distribution. If the model preserves good prediction accuracy, we say the model is generalizing well to the samples in the task distribution. However, training an ML model to reach the desired performance could not be easy, and sometimes it may require tuning its hyperparameters. Hyperparameters are adjustable values that can be tuned to obtain training with good performance. Tuning the hyperparameters is a consistent part of training the models, as their choice might significantly affect the optimization result. If the model's performance is unsatisfactory at validation time, then hyperparameters can be tuned to improve it.
Finally, the model performance is tested again on a test dataset $\mathcal{T}$, never seen during training or validation, to assess whether the model can generalize to samples that may have to classify once it has been deployed. Performance on the test data is then the final and more reliable evaluation to understand the goodness of the trained model.

Neural Networks. A Neural Network (NN) is an information processing paradigm inspired by how biological nervous systems, such as the brain, process information. The pivotal element of this

Table 2.1: *Neural network topologies.*

Feedforward only allow signals to travel one way only: from input to output. There is no feedback (loops). The output of any layer does not affect that same layer. This type of organization is also referred to as bottom-up or top-down.	**Feedback networks** (*or Recurrent networks*) can have signals traveling in both directions by introducing loops in the network. Feedback networks are powerful and can get extremely complicated.
Fully connected has each neuron connected to every neuron in the previous layer, and each connection has its weight.	**Sparsely connected** has fewer links than the possible maximum number of links within that network.
Single layer	**Multilayer**

paradigm is the new structure of the information processing system. It contains numerous highly interconnected processing elements, namely **neurons**, working in unison to solve specific tasks. Although neural networks appear to be a recent development, the first model was introduced in 1943, i.e., the McCulloch and Pitts Model [188]. The McCulloch-Pitts (MP) neuron is a simple process unit modeled as a binary threshold unit that can implement only linearly separable functions. The unit fires if the net input $\sum_j w_j I_j$ reaches (or exceeds) the unit's threshold T:

$$\hat{y} = g\left(\sum_j w_j I_j - T\right) \qquad g(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

The weight w_j represents the strength between neuron j and the j-th input feature. The function $g(x)$ is also called **activation function** and determine whether the neurons fire (is active) or not. Among the most common ones, we find the Rectified Linear Unit (ReLU), which returns the maximum between the neuron activation and 0.

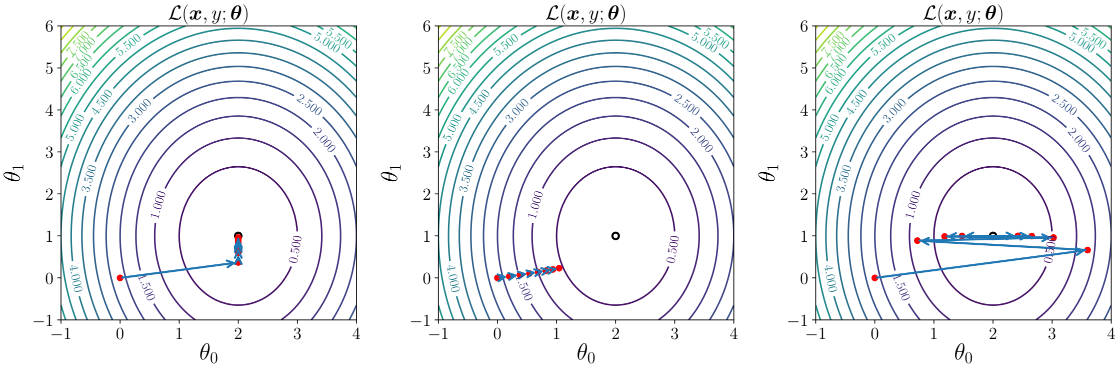
$$g(x) = \max(0, x)$$

Combining multiples MP neurons allows us to derive more complex network topologies, and their main differences are presented in Table 2.1.
A single layer represents the structure of a perceptron, which can implement only linearly separable functions. Conversely, multi-layer networks improve their approximation performance by stacking additional *hidden layers*.

Shallow vs Deep Networks. From the *universal approximation theorem* we know that a two-layer network (one input layer, one hidden layer) with a suitable activation function can approximate any continuous function. This statement, however, does not give us information about how large this network should be. Before the advent of neural networks, this theorem was used to develop shallow networks composed of a few large hidden layers, increasing the number of parameters significantly. Deep learning has significantly revolutionized the approach with which we now design new ML architectures. Deep networks simulate the brain's behavior, in which the electric signals propagate across different layers. A deep network tries to approximate the target function by stacking multiple non-linear hidden layers but decreasing the number of required parameters, as they are less wide.

Backpropagation Learning Algorithm Backpropagation is an algorithm for learning the weights in neural networks. The algorithm is based on gradient descent and can be seen as a greedy algorithm where the gradient gives information about the best path to follow at each step to increase/decrease our objective function. It is a first-order iterative optimization algorithm for finding the minimum of a function. It can be defined as:

$$\boldsymbol{\theta}_{ji}^{(t+1)} \leftarrow \boldsymbol{\theta}_{ji}^{(t)} - \eta \frac{\partial \ell}{\partial \boldsymbol{\theta}_{ji}^{(t)}}$$



(a) *Perfect η.* **(b)** *Small η.* **(c)** *Large η.*

Fig. 2.1: *Importance of tuning the learning rate size η in the optimization of θ. By choosing a correct value (2.1a) the gradient updates bring the optimized variable to the minimum of the loss. When the learning rate is too low (2.1b), the updates for θ are insufficient to reach the optimum. When the learning rate is too big (2.1c), the optimized variable overshoots the minimum and can not descend properly.*

where η is the learning rate, $\boldsymbol{\theta}_{ji}$ are the model's weights updated at each step, and ℓ is the loss function we optimize during training. Intuitively, we could say that we determine the gradient direction, then move in the opposite direction since we want to minimize the error. In the backpropagation algorithm, the input of the network is propagated layer after layer in the forward direction (*forward pass*), and finally, the error/loss made by the network is propagated backward and weights to properly update them (*backward pass*). This optimization approach is not guaranteed to find a globally optimal solution, but it works well in finding local optima.

One of the most challenging problems for backpropagation is finding a good η (learning rate), which can influence the algorithm's convergence. If η is small, the algorithm will converge very slowly; however, when it is too big, there will be an oscillating problem that will not bring the algorithm to convergence. In Fig. 2.1 image, we can see the difference in the convergence between the same algorithm run with different values of η. We can observe that the first case is too slow to reach the minimum, while the oscillation becomes smaller in the second and third cases. Finally, the algorithm will not converge in the last case as the oscillation becomes larger and larger. One possible remedy to this problem is the adoption of the **momentum term** [221], a simple heuristic that can help find a good value for η. The momentum term introduces a dependency on the previous step, avoiding or mitigating the oscillatory phenomena. The obvious disadvantage is the need to set two parameters instead of one.

Overfitting and Underfitting. Overfitting and underfitting are the two common phenomena in ML that lead to poor performance (see Fig. 2.2). Underfitting occurs due to the low-capacity model, which can not capture the relationship between the input examples and the output labels, leading the model to perform poorly on the training data. To reduce underfitting, the model capacity can be expanded (e.g., us- ing more neurons in one layer, tuning the hyperparameters, etc.). However, increasing the model's capacity may lead to overfitting. Overfitting is indeed the result of the over-capacity of the model and over-minimizing the error on the training set. Overfitted model performs well on the training data but does not perform well on the validation or test data. This is because the model memorizes the training data and can not generalize to unseen examples. A straightforward solution to reduce overfitting is early-stopping, which stops the learning procedure when the minimum validation set error is reached. At this point, the net should be generalizing in the best possible way. When learning is not stopped, "overtraining" occurs, and the performance of the net on the dataset as a whole decrease, even though the error on the training data still gets smaller. However, due to the relevance of the overfitting issue, more approaches have been developed in the literature, such as (i) pruning [167], which reduces the model's complexity

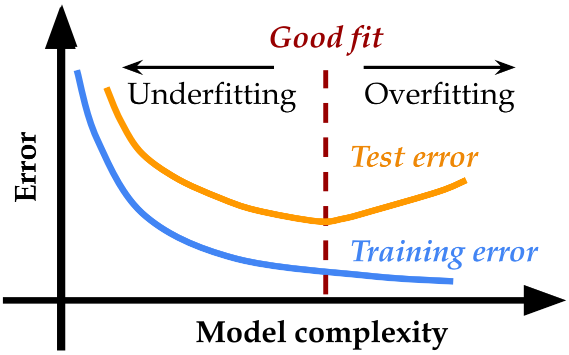


Fig. 2.2: *Relationship between model complexity and training/test error.*

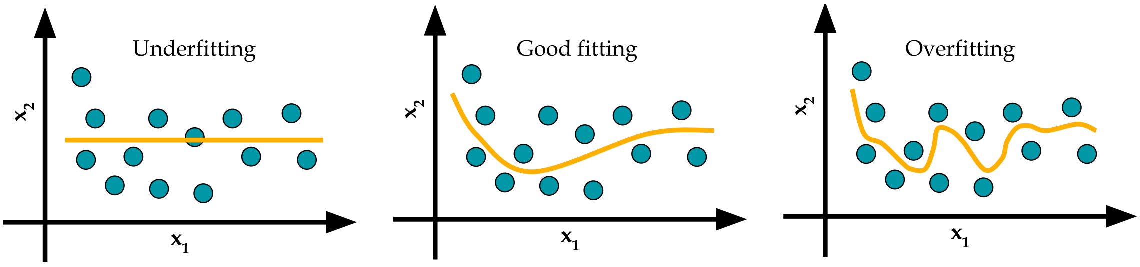


Fig. 2.3: *Examples of models in case of underfitting, good fitting, and overfitting.*

by removing low-weight connections and then retrains the model to recover its performance; (ii) regularization [149], which aims to bound and control the model's weights; (iii) dropout [252], which randomly removes some neurons at different epochs during training; (iv) data augmentation [244], which artificially enlarge the dataset using label-preserving transformations; or (v) dropout, which randomly removes some neurons at different epochs during training. In Fig. 2.3 we show the decision function of two distinct models affected by underfitting and overfitting. Moreover, we relate the model complexity with the test error showing that these two effects can influence the model before and after a certain threshold.

Convex Learners. However, when dealing with a limited dataset, including only a few samples per class, NNs may perform extremely poorly. In the following, we present other ML models that we call convex learners as they optimize convex objectives that can achieve acceptable levels of accuracy even when the data is fairly scarce. We study convex learners also due to their broad usage in industry [260], derived from their low computational cost, excellent results, and good interpretability [69; 267]. We thus present Logistic Regression Classifier (LC), Ridge Classifier (RC), and Support Vector Machine (SVM).
Logistic [124] and Ridge classifiers [121] are linear regression frameworks that can also be used for binary classification tasks. A logistic function has a general form like this:

$$y_i = \frac{1}{1 + e^{-(\boldsymbol{\theta}_0 + \boldsymbol{\theta}_1 x_{1,i} + \cdots + \boldsymbol{\theta}_d x_{d,i})}}$$

This form corresponds to a single-layer neural network where $\boldsymbol{\theta}$ are the model's coefficient, and $x_{d,i}$ is the j-th feature of samples i. The linear decision boundary is a consequence of the structure of the regression function and the use of a threshold in the function to classify. Ridge classifier adopts a "shrinkage" method to reduce or shrink the coefficients in the resulting regression. This reduces the model's variance, avoiding falling under the overfitting regime.
Support Vector Machine (SVM) [66] is a mighty and less computationally expensive linear classi-

fication algorithm. We can formally define the SVM linear classifier in this way:

$$\mathcal{M}(x, \boldsymbol{w}, b) = g(\boldsymbol{w}^T \boldsymbol{x} + b) \qquad g(z) = \begin{cases} 1 & \text{if} \quad z \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

where $w^T x + b$ represents a **separating hyperplane** for the two classes, and $g(z)$ is the decision function which returns the predicted class for the observation x. Unlike other linear ML models, SVM finds a separating hyperplane that maximizes the margin between class samples. The resulting hyperplane is considered the best since it separates classes better than all possible hyperplanes. Intuitively, the farthest the boundary is from the points, the greater the prediction's confidence. Moreover, SVM formulation corresponds to the optimization problem of a convex function with only linear constraints, for which a unique minimum solution corresponds to the optimal margin classifier exists. The decision boundary depends on a small number of points, namely "support vectors", whose distance to the separating hyperplane is equal to the minimum margin from the decision boundary.

Although the above presented convex learners are designed to work for linearly separable data, SVMs can be extended to encompass also non-linear tasks. SVMs allow the usage of a strategy, namely kernel trick, for learning a separating hyperplane in a new space where data are likely to be linearly separable.

2.1.2 Unsupervised Learning

Unsupervised learning is a paradigm of the machine learning field which is based on the training of knowledge without using a teacher. Unsupervised algorithms learn from unlabeled data; hence they learn the hidden structures within them. One of the most common applications in unsupervised learning is **clustering**. Clustering consists of estimating how data are organized in the space such that they can reconstruct the prior probability distribution of data (see Fig. 2.4). In other words, they are used with the goal of grouping a set of objects in such a way that objects in the same cluster are strongly similar (*internal criterion*) and objects from distinct clusters are strongly dissimilar (*external criterion*). Several clustering algorithms have been offered in the literature, and each manages data differently. Some of the clustering algorithms we will test against adversarial noise in Chapter 7 are K-Means[115], Spectral [275] and Hierarchical [199] clustering.

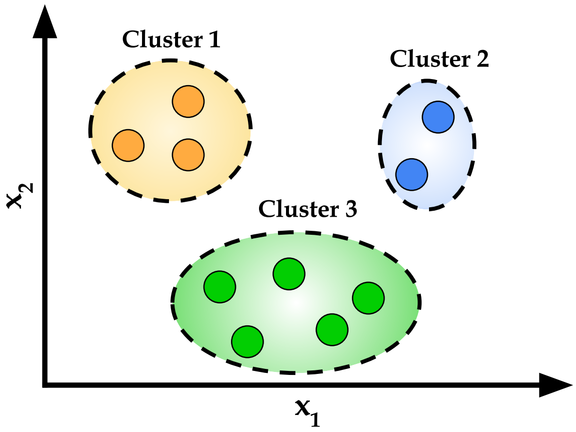


Fig. 2.4: *Effect of clustering on a 2-dimensional space.*

K-Means Clustering Partitioning clustering algorithms separate the data set into the specified number of clusters based on the similarity or distance among the data samples. Among them, K-Means [115] is one of the simplest, most famous, and most used iterative clustering algorithms [285]. Being a partitioning algorithm, it aims to partition n objects into K maximal cohesive groups.

Algorithm 1: K-Means clustering algorithm.

```
Input: $\boldsymbol{X}$, $K$
Output: $\mathcal{S}$

initialize cluster centroids $\mathcal{S} = \{s_1, s_2, ..., s_k\}$
repeat
    $\forall_i \in \boldsymbol{X} \quad s^{(i)} \in \arg\min_j \|\boldsymbol{x}_i - c_j\|^2$        // partition data

    $\forall_j \in \mathcal{S} \quad \boldsymbol{c}_j = \frac{\sum_{\boldsymbol{x}_i \in s_j} \boldsymbol{x}_i}{\sum_{\boldsymbol{x}_i \in s_j} 1}$        // update centroids
until All points remain unchanged (convergence)
return $\mathcal{S}$
```

Algorithm 2: Spectral clustering algorithm.

```
Input: $\boldsymbol{Z}$, $K$
Output: $\mathcal{S}$
Let $L$ be the normalized graph Laplacian associated to $\boldsymbol{Z}$.
Compute the $K$ smallest eigenvectors $v_1, \dots, v_K$ of $L$
Let $\boldsymbol{V} = [v_1, \dots, v_K] \in \mathbb{R}^{n \times K}$ the resulting data embedding when using $v_1, \dots, v_k$ as
  features
Form the matrix $\boldsymbol{U} \in \mathbb{R}^{n \times K}$ from $V$ by normalizing the row sums to have norm 1 that is:
    $$u_{ij} = \frac{v_{ij}}{\left(\sum_{k=1}^{K} v_{ik}^2\right)^{1/2}}$$
Run the K-Means algorithm on $\boldsymbol{U}$ to find clusters $\{s_1, \dots, s_K\} \in \mathcal{S}$.
return $\mathcal{S}$
```

The number of clusters K needs to be predefined before the execution. It aims to partition n objects into K maximal cohesive groups. Each sample is associated with the cluster having the nearest center (centroid). Mathematically speaking, given a set of samples $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n\}$, partition it into K clusters $\mathcal{S} = \{s_1, \dots, s_K\}$ with the corresponding centroids $\mathcal{C} = \{\boldsymbol{c}_1, \dots, \boldsymbol{c}_K\}$ so as to minimize the sum of internal-cluster variances:

$$\mathcal{S} \in \arg\min \sum_{j=1}^{K} \sum_{\boldsymbol{x}_i \in s_j} \|\boldsymbol{x}_i - \boldsymbol{c}_j\|^2$$

The K-Means algorithm to optimize the above objective function is given in Alg. 1. Initially, K-Means randomly chooses K samples as centroids and find the nearest data points of the chosen centroids to form K clusters. Then, it iteratively updates the centroids for each cluster until the algorithm converges. Convergence is reached after a fixed amount of iterations or when the centroids are no anymore updated significantly. This algorithm is guaranteed to converge – but not to the optimal solution. Like gradient-descent methods, the K-Means algorithm is guaranteed to converge in a finite number of steps to a local minimum. Moreover, it is a polynomial algorithm: $\mathcal{O}(Kn)$ for assigning each sample to the closest cluster and $\mathcal{O}(n)$ to update the centroids of the clusters. In conclusion, K-Means is a straightforward and efficient method, but, on the other hand, it is strongly sensible for the initialization phase. If the initial centroids are not chosen correctly, the algorithm converges to a local minimum of the error function. Therefore, the results can differ from one execution and another, lacking inconsistency.

Algorithm 3: Hiearchical clustering algorithm.

```
Input: $\boldsymbol{X}$, $\mathcal{J}$
Output: $\mathcal{S}$, $H$
Let $H = \{\{\boldsymbol{x}_i\}, \ldots, \{\boldsymbol{x}_n\}\}$ be set of singleton clusters formed with samples in $\boldsymbol{X}$
repeat
    Find pair of clusters $s_i, s_j \in H$ such that
        $s_i, s_j \in \arg\max_{i \neq j} \mathcal{J}(s_i, s_j)$
    Update $H$ by merging $s_i$, $s_j$
        $H = H \backslash s_i \backslash s_j \cup \{s_i \cup s_j\}$
until There is only one cluster left, i.e., $|H| = 1$
Extract $\mathcal{S}$ from $H$ according to the cutting threshold policy
return $\mathcal{S}$, $H$
```

Spectral Clustering With the K-Means algorithm, cluster membership points can be assumed to be in a spherical area since the centroid is typically in the center of the cluster. This assumption is invalid in the presence of non-convex shapes, and therefore, K-Means may not work well. Spectral clustering [275] algorithm exploits the graph theory to cluster objects that are connected but not necessarily compact or clustered within convex boundaries. The algorithm has in input a similarity graph $\boldsymbol{Z} \in \mathbb{R}^{n \times n}$ and the K number of clusters to construct. The similarity graph $\boldsymbol{Z}$ maintains the similarity scores between data nodes, specifically Z_{ij} refers to the similarity between samples i and j in the data $\boldsymbol{X}$. The performance of spectral clustering relies heavily on the similarity measure used to construct the similarity graphs. Among the vast plethora of similarity measures, we find cosine similarity, Pearson correlation coefficient, or inverse distance-based, which are the ones we also investigate in this thesis. Once the similarity matrix has been defined, the associated Laplacian matrix is calculated, and the K smallest eigenvectors are extracted. Finally, these eigenvectors are used as features, and the K-Means algorithm is applied to them for extrapolating the K clusters. The overall algorithm is reported in Alg. 2.

Applying K-Means to Laplacian eigenvectors allows us to find a cluster with non-convex boundaries. Moreover, embed data points in spectral embedding space, where the clusters are more pronounced. However, it can be computationally expensive since the algorithm must compute eigenvalues and eigenvectors and perform clustering over them. Indeed, its time and space complexity are respectively $\mathcal{O}(n^3)$ and $\mathcal{O}(n^2)$, n being the number of data samples, which implies more difficulties scaling for large datasets.

Hierarchical Clustering. Hierarchical clustering [199] belongs to the category of agglomerative clustering algorithms, where clusters are composed in a bottom-up manner. Conversely to the partitioning clustering algorithms (e.g., K-Means, Spectral), which produce single data partitions, Hierarchical clustering creates a structure, such as a dendrogram, that depict the hierarchy of clusters. Firstly, the algorithm creates a singleton cluster for every input object. Then it iteratively merges the two most similar clusters into a single cluster. This procedure is repeated until there is only one cluster left. The result is a tree structure where the root is the last cluster containing all the samples, and the leaves are the singleton input data. Finally, a dendrogram showing the clusters tree structures and their distance are used to define the cutting threshold. Cutting the dendrogram at a specific similarity score would create a set of clusters where each cluster is the root of a tree structure, and each leaf is a single cluster (see Fig. 2.5). Moreover, compared to K-Means and Spectral clustering, the number of clusters K is not necessarily required to be specified, as we can use a minimum similarity score between clusters as a threshold and have a non-predefined number of clusters.

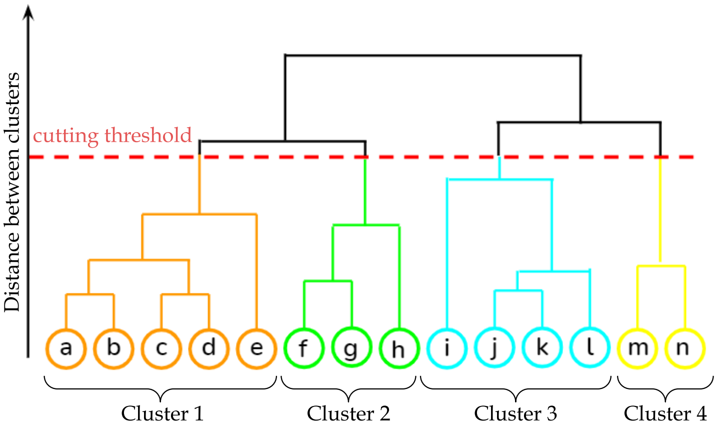


Fig. 2.5: *A dendrogram representing the hierarchical clustering algorithm. The cutting threshold has originated in four distinct clusters of objects.*

The pseudocode of Hierarchical clustering is given in Alg. 3. The input now is the data samples $\boldsymbol{X}$, and $\mathcal{J}$ is a measure used to evaluate the distance or similarity between two clustering partitions. The most common similarity measures for $\mathcal{J}$ are single-linkage, complete-linkage, average-linkage, and Ward's linkage. With single-linkage (complete-linkage) the distance between two clusters is defined as the minimum (maximum) distance between any sample in the first cluster and another one in the second cluster. Average-linkage calculates the distance of all data samples from the first cluster with all others from the second cluster and takes the average distance as the distance between the two clusters. Ward-linkage analyzes the variance of the two clusters when merged. Beyond them, as for Spectral clustering, the similarity measure can be ad-hoc customized for the specific task. Compared with Spectral clustering, Hierarchical clustering has a relatively smaller time complexity $\mathcal{O}(n^2 \log n)$, n being the number of data samples. However, its computational complexity is higher than that of K-Means, which is linear.

2.2 Adversarial Machine Learning

The extensive use of machine learning in safety-critical systems has raised questions about their robustness in the presence of malicious users. In recent years, researchers have gained soaring interest in investigating ML applications' vulnerabilities because of the disastrous consequence they may cause. For example, ML models are adopted in intrusion detection systems to prevent harm from malicious intrusion by detecting adversarial traffic. An attacker, intending to bypass detection, seeks to tamper with any component of the target system, including ML components used to make decisions or their data, to reduce the quality of detection. Similarly, for spam detection, where it has been shown that ML can be easily confused if a malicious spammer mixes spam content with ham-like messages. Due to the natural adversarial nature of these applications, identifying and protecting their attack vectors is mandatory for testing the robustness of such systems. Adversarial machine learning is concerned thus with testing the robustness and protecting ML systems from intentional attacks to preserve their availability, integrity, and confidentiality. Availability attacks are when the attacker attempts to prevent legitimate users from accessing the model, either by decreasing its accuracy and thus obtaining meaningless predictions or by causing a Denial-of-Service (DoS) of the system by overloading it with requests. Attacks on integrity as those that induce the model towards target outputs or behaviors previously chosen by the attacker. Finally, attacks on confidentiality attempt to expose the model structure or parameters or the data used to train and validate the model. This issue is recently emerging fast for competitive and legal reasons. ML models can be valuable intellectual property companies want to preserve to maintain a competitive advantage over competitors. Even more severe, data often includes

	Attack Violation		
Attack Influence	**Integrity**	**Availability**	**Privacy/Confidentiality**
Exploratory	Evasion (adversarial examples)	Energy-latency attacks	Model extraction / stealing and model inversion
Causative	Poisoning (*targeted, or backdoor*)	Poisoning (*indiscriminate, sponge*)	-

Table 2.2: *Categorization of attacks against machine learning depending on the attacker's target violation and attack influence. In red the attacks subject of this thesis.*

personal and private information (e.g., patient data), the use of which is governed by current data privacy guidelines, such as the EU's General Data Protection Regulation (GDPR). Thus, protecting against such attacks also demands compliance with Governments' laws and regularization.

The big breakthrough in this domain was the discovery that ML is extending the attack surface, i.e., all the possible attack vectors that represent the system's vulnerabilities, both at training time or after the model's deployment in a production application, i.e., test time. In 2010 Barreno et al. [17], proposed a framework to categorize such attacks against machine learning, identifying two potential influence attacks that can deceive learning systems, i.e., *exploratory* and *causative* attacks. Table 2.2 summarizes the taxonomy of malicious threats against ML systems and highlights the main topic of this thesis.

2.2.1 Exploratory Attacks

Exploratory attacks exploit the target system's weaknesses at test time to reach the attacker's goal.

Evasion Attacks. Among them *evasion attacks* have been getting a lot of attention in recent years [28; 105; 255; 261]. In such attacks, the attacker alters malicious input data, namely adversarial examples, to have them misclassified by the model. Even more worrying is that even a wisely crafted but imperceptible noise injected in input samples may lead the ML systems to output wrong predictions, threatening the trustworthiness of ML systems. In Fig. 2.6 we give an example of a parrot adversarial example. The model correctly classifies the input sample shown on the left. However, when the input is slightly perturbed, but still looks like a parrot for humans, it is misclassified as a dog. Injecting the adversarial noise has moved the input sample from the decision boundary of parrots (orange) toward the decision region of dogs (blue). The attacker succeeds in their goal as the adversarial parrot image has misled the ML model with high confidence. Beyond this toy example, adversarial examples have been exploited in more realistic scenarios. For example, an attacker may add a sticker [91] to a stop sign to have it misclassified as another road sign, potentially causing a collision. Or slightly alter a malware to have it misclassified by antivirus as a legitimate application [71]. Alternatively, wearing a pair of adversarial glasses to elude surveillance controls. Sharif et al. [240] have shown that an attacker can evade facial biometric systems, widely used in surveillance and access control, by wearing wisely crafted accessories. The presence of one of these accessories allows the attacker to impersonate another trusted individual (see Fig. 2.7).

Traffic sign recognition, malware detection, and biometric access control are just a few of the domains where evasion attacks have seriously posed questions about their trustworthiness in practical applications. The primary responsibility is that ML algorithms have been designed carelessly for the security aspect required for relying on them in real-world applications. This paradigm nowadays is less remarked on, and novel defensive techniques to protect ML from these attacks have been developed [20; 152; 185; 232; 235; 161; 65; 310]. Unfortunately, these often prove ineffective [37], and the search for a robust model against adversarial examples is still ongoing.

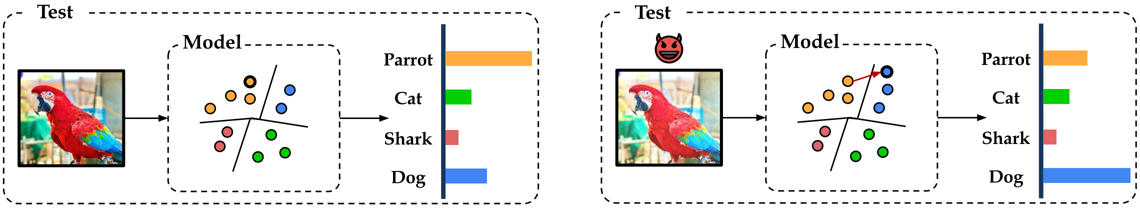


Fig. 2.6: *Example of an evasion attack on the animal classifier. On the left, the expected behavior of the model, which correctly classifies the input image. On the right, the adversarial example capable of misleading the ML model. The red arrow indicates how the original input parrot was moved into feature space, crossing the model's decision boundary.*

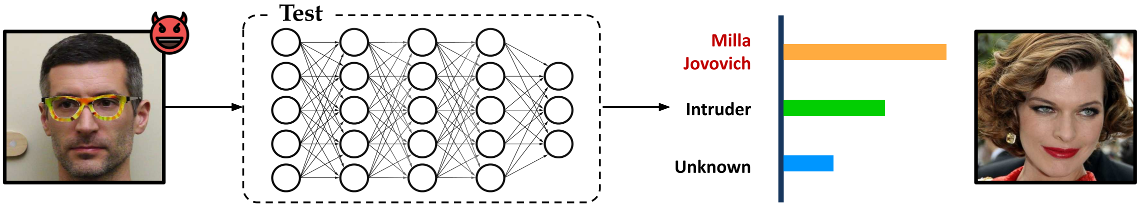


Fig. 2.7: *Example of evasion attack against biometric control system based on ML. Wearing a pair of earrings allows the intruder to be classified as Mila Jovovich.*

Sponge Attacks. More recently also *sponge attacks* have been investigated by Shumailov et al. [245] to increase the energy consumption of ML models. This attack aims at draining the system's batteries faster, compromising its availability to legitimate users. Because this is a recent threat, no defenses against it have been evaluated. One possible strategy would be to adopt an anomaly detector that tracks user requests and slows down or blocks users whose requests are more energy demanding. More details about sponge examples are given in Chapter 6 when presenting our energy-latency attack.

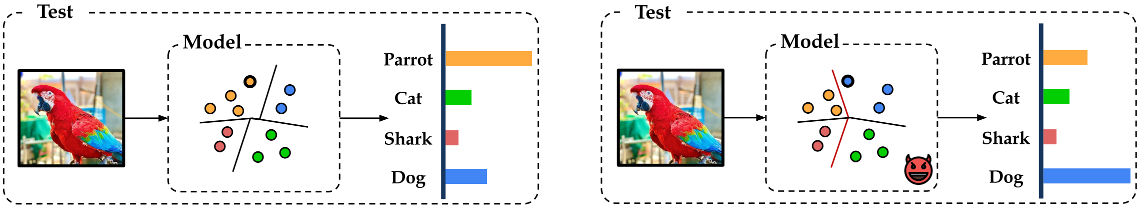


Fig. 2.8: *Example of a poisoning attack on the animal classifier. On the left, a clean model that behaves as expected, i.e., it correctly classifies the input image. On the right, the poisoned ML model misclassifies the pristine input as desired by the attacker. The red decision boundary indicates how the model has changed after being targeted by poisoning.*

2.2.2 Causative Attacks

Causative attacks influence the learning process by tampering with the *training data* or *model* to meet the attacker's objective once the model is deployed. Causative attacks are helpful especially in applications where the attacker can not craft an adversarial perturbation for each test sample, for example, when the attack must be real-time. Fig. 2.8 depicts the influence of a poisoning attack on a ML model decision boundary. Specifically, compared to the pristine model (left), the poisoned one (right) has a different decision boundary that now meets the attacker's goal. Compared to the evasion attack seen in Fig. 2.6, the attacker's effort was made during training by opening a backdoor while the victim was collecting the training data or training the model. At test time, the attacker exploits the opened vulnerability without optimizing/manipulating the input image.

Data Poisoning Attacks. Among the most famous and explored attacks *data poisoning* are the more historical ones, to which the ML security literature has also paid the most attention. These attacks assume the attacker's capacity to influence learning by manipulating training samples. The attacker acts only during the data gathering phase, e.g., compromising local data that are subsequently shared to a remote central node that aggregates distributed data for training the model. Being *data poisoning* attacks the main subject of this thesis, and for which a vast plethora of works exist, we deeply investigate this topic in Chapter 3. The ever-increasing interest in *data poisoning* threat has indeed brought the development of many papers. However, proper categorization of them based on the attacker's and defender's assumptions is missing or incomplete, leading to unfair comparison, misconceptions, or a false sense of insecurity. For that reason we proposed two papers around this categorization, i.e., [61; 62], used as bases for Chapter 3.

Model Poisoning Attacks. *Model poisoning* attacks are a more recent causative threat vector against ML models. In contrast to data poisoning, the attacker directly manipulates their local model or gradient updates without the need to modify data or labels. We further investigate *model poisoning* attacks in Chapter 6 when proposing our model poisoning attack to increase energy consumption [61].

Chapter 3

Machine Learning Security against Data Poisoning

Research Question #1

How can we categorize/distinguish poisoning attacks against ML?

Poisoning attacks are staged at training time by manipulating the training data or compromising the learning process to degrade the model's performance at test time. Among the two scenarios, the case where data are influenced by the attacker, namely data poisoning, has attracted increasing attention from ML stakeholders, perhaps after the incident of Tay [156] (see Sec. 3.1), to the point that now it is considered the largest concern for ML applications [107; 151]. We identified three main categories of data poisoning attacks [62; 63], namely indiscriminate, targeted, and backdoor poisoning attacks.

Indiscriminate poisoning attacks are staged to maximize the classification error of the model on the (clean) test samples. The attacker aims to reduce the system's availability to legitimate users who can not trust the output of the poisoned model.

Targeted poisoning attacks influence the model to cause misclassification only for a specific set of (clean) test samples.

In *backdoor* poisoning attacks, the training data is manipulated by adding poisoning samples containing a specific pattern, referred to as the backdoor trigger, and labeled with an attacker-chosen class label. This typically induces the model to learn a strong correlation between the backdoor trigger and the attacker-chosen class label. Accordingly, the input samples that embed the trigger are misclassified at test time as samples of the attacker-chosen class, while the pristine samples remain correctly classified.

However, outside of our two works and a few others, this distinction between different types of poisoning and the underlying assumptions is not evident in the literature, often leading to confusion between target and backdoor attacks. This Chapter present our works [62; 63] that explores the literature around data poisoning attacks to categorize them according to their threat model and identify possible defenses and countermeasure to counter them.

Underlying Problem. We argue that the literature around poisoning is often quite chaotic. The distinction between the different poisoning attack types is unclear, leading to unfair comparisons in the experimental evaluation. In some works, for example, we can see comparisons between attacks with essentially different goals (e.g., targeted compared to backdoor attacks) and assumptions (e.g., attacks manipulating a few data vs. attacks on the entire training set) that require different evaluations. Moreover, perhaps due to the chaotic scenario, some works are introducing methodological novelty in staging poisoning attacks, but their applicability remains unknown.

Proper categorization of existing attacks based on their threat models is therefore demanding for shedding light on that field to overcome the difficulties mentioned above.

Related Work. Differently from existing surveys in the literature on ML security, which either consider a high-level overview of the whole spectrum of attacks on ML [24; 42] or are specific to an application domain [257; 291], our work [63] focuses solely on data poisoning attacks and defenses, providing a greater level of detail and a more specific taxonomy. Other survey papers on poisoning attacks do only consider backdoor attacks [98; 139; 162], except for the work by Goldblum et al. [104]. Compared to the latter, our survey categorizes attacks and defenses based on a more systematic threat modeling, introduces a unified optimization framework for poisoning attacks, matches the defenses with the corresponding attacks they prevent, and discusses the historical timeline of poisoning attacks since the early developments in cybersecurity applications of ML, dating back to more than 15 years ago.

Contributions and Outline. In this Chapter, we examine the contributions proposed in our survey [63] and magazine [62] papers, aimed at analyzing the literature on poisoning attacks, their real-world implications, and possible countermeasures to stop them. We will focus more on the parts where the author of this thesis contributed the most, i.e., formalization of the threat model for attacks and matching with defenses. Further details on the defenses against poisoning, mostly elaborated by other co-authors, are given in the two papers.

In our works, we identified three main categories of data poisoning attacks [62; 63]. These include indiscriminate, targeted, and backdoor poisoning attacks, each of which assumes the capacity of the attacker to tamper with the training data to reach the attacker's goal at test time. *Indiscriminate* poisoning attacks are staged to maximize the classification error of the model on the (clean) test samples. The attacker aims to reduce the system's availability to legitimate users who can not trust the output of the poisoned model. *Targeted* poisoning attacks aim to influence the model to cause misclassification only for a specific set of (clean) test samples. In *backdoor* poisoning attacks, the training data is manipulated by adding poisoning samples containing a specific pattern, referred to as the backdoor trigger, and labeled with an attacker-chosen class label. This typically induces the model to learn a strong correlation between the backdoor trigger and the attacker-chosen class label. Accordingly, the input samples that embed the trigger are misclassified at test time as samples of the attacker-chosen class. In the last two scenarios, targeted and backdoor attacks, the model's performance is not compromised except for target points known only to the attacker, thus making it more difficult for the victim to stop the attack.

Specifically, (i) we identify in Sec. 3.1 the main practical scenarios that enable staging such attacks on ML models (ii) we provide in Sec. 3.2 a comprehensive framework for threat modeling of poisoning attacks and categorizing defenses, responding to our research question #1, i.e., "how can we categorize/distinguish poisoning attacks against ML?" (iii) we use our framework to categorize more than 50 papers on poisoning attacks, and we derive a unified formalization for their optimization in Sec. 3.3; (iv) we take advantage of our framework to match specific attacks with appropriate defenses according to their strategies in Sec. 3.4; (iv) we review in Sec. 3.5 the other domains where poisoning have been investigated; and (v) we finally conclude this Chapter in Sec. 3.6 to discuss the limitation of our work and propose future development.

3.1 Poisoning in Real-World Applications

Suppose you call one of your company suppliers to understand why they stopped emailing you about the month's promotions. The supplier replies that they continue to send their promotion as usual and invite you to check the spam. The supplier was right! The emails ended up in your spam folder, together with other communications from that company. Be aware that this could not have happened by accident but as a result of fraud, in which an evil competitor ensures that the email client marks any email from the victim company as spam. To this end, this malicious company could flood you with spam containing the victim company's name - until the ML-based spam filter associates this benign name with the property “spam”, thus trashing future promotions. This scenario is an instance of a machine learning security threat called data poisoning, described in 2008 by Nelson et al. [200], taken as an example in [62]. Under this setting, malicious users may cause failures in ML systems (e.g., spam filters) by tampering with their training data, thereby posing real concerns about the trustworthiness of the overall application. Moreover, because of the opacity of ML models, it is even more challenging to properly test them during their lifetime, identifying possible vulnerabilities or attacks before malicious users exploit them. For this reason, the European Union (EU) has recently approved a set of ethical guidelines for developing *trustworthy* ML and artificial intelligence (AI) algorithms, and the so-called EU AI Act [89] (see also Pelillo et al. [216] for an interdisciplinary perspective on these issues). These regulations require such systems to provide accurate and human-aligned decisions, which follow the principles of being explainable, fair, robust, and accountable.

Unfortunately, the road toward developing trustworthy AI/ML systems is paved with many obstacles. In particular, it is not only a problem of designing the training algorithms or ML architectures *right*. The data, specifically the data gathering process, plays a crucial role too. As Gary McGraw says, “it matters just as much as the rest of the technology, probably more”. While data can be a strength for AI/ML models, it may also be their most vulnerable Achilles’ heel. Mindful monitoring of the data collection procedure is becoming imperative, especially after the latest incident in real-world applications. In 2016 Tay, an artificially intelligent chatbot developed by Microsoft’s Technology and Research, started to twit inappropriate messages because of a poisoning attack [156]. Due to the large amount of data that ML systems consume, it is quite tricky to identify poisoning samples in a massive training set, leading Microsoft to switch off the service. Chatbots in other languages have shared its fate, including a Chinese [56] and a Korean [146] version. Another attack showed how to poison the auto-complete feature in search engines [8]. Finally, a group of extremists submitted wrongly-labeled images of portable ovens with wheels tagging them as *Jewish baby strollers* to poison Google’s image search [106]. Several sources confirm that poisoning is already carried out in practice [107; 189; 151].

Due to their practical relevance, various scientific articles have been published on training-time attacks against ML models. Our survey [63] covers the literature on poisoning on supervised classification models in the computer vision domain, where the vast majority of work has been done. However, we want to remark that data poisoning has been investigated earlier in other application domains, briefly investigated in Sect. 3.5.

3.2 Modeling Poisoning Attacks and Defenses

The literature and efforts toward the poisoning problem have increased in recent years, developing attacks and defenses. In fact, on the attack side, we study how a possible attacker could be capable of compromising the ML training phase and what vulnerabilities these models expose. These are all analyses that serve to test the robustness of AI/ML models. On the defense side, new detection or sanitization strategies are being developed on both data and models to remove or reduce the impact of potential poisoning attacks.

In the remainder of this section, we investigate the attacks and defenses modeling frameworks we proposed in [63]. The two frameworks model the profile of attackers, who wish to harm the system, and defenders, who wish to protect or repair the system, describing their motivation and capabilities. We discuss how we categorize poisoning attacks against learning models, answering our research question #1, and how defenses try to mitigate them. In doing this, we revisit the framework by Muñoz-González et al. [197] to systematize poisoning attacks according to the attacker’s goal, knowledge of the target system, and capability of manipulating the input data or model. We similarly conclude by characterizing the defender’s goal, knowledge, capability, and strategy to mitigate the impact of poisoning attacks. The notation and symbols used throughout this Chapter are summarized on page v.

3.2.1 Learning Settings

We define here the three main scenarios under which ML models can be trained, and which can pose serious concerns in relation to data poisoning attacks. We refer to them below respectively as (i) *training from scratch*, (ii) *fine tuning*, and (iii) *model training.* In Fig. 3.1, we conceptually represent these settings, along with the entry points of the attack surface that enable staging a poisoning attack.

Training from Scratch (TS) and Fine Tuning (FT) In the *training-from-scratch* and *fine-tuning* scenarios, the user controls the training process, but collects the training data from external repositories, potentially compromised by attackers. In practice, these are the cases where data gathering and labeling represent time-consuming and expensive tasks that not all organizations and individuals can afford, forcing them to collect data from untrusted external sources. The distinction between the two scenarios hinges on how the collected data are employed during training. In the *training-from-scratch* scenario, the collected data is used to train the model from a random initialization of its weights. In the *fine-tuning* setting, instead, a pretrained model is typically downloaded from an untrusted source, and used to map the input samples on a given representation space induced by a feature mapping function ϕ. Then, a classification function f is fine tuned on top of the given representation ϕ.

Model Training (MT) In the *model-training* (outsourcing) scenario, the user is supposed to have limited computational capacities and outsources the whole training procedure to an untrusted third party, while providing the training dataset. The resulting model can then be provided either as an online service which the user can access via queries, or given directly to the user. In this case, both the feature mapping ϕ and the classification function f are trained by the attacker (i.e., the untrusted party). The user, however, can validate the model's accuracy on a separate validation dataset to ensure that the model meets the desired performance requirements.

3.2.2 Attack Framework

Attacker's Goal. The goal of a poisoning attack can be defined in terms of the intended security violation, and the attack and error specificity, as detailed below.

`Security Violation.` It defines the security violation caused by the attack, which can be: (i) an *integrity* violation, if malicious activities evade detection without compromising normal system operation; (ii) an *availability* violation, if normal system functionality is compromised, causing a denial of service for legitimate users; or (iii) a *privacy* violation, if the attacker aims to obtain private information about the system itself, its users, or its data.

`Attack Specificity.` It determines which samples are subject to the attack. It can be: (i) *sample-specific* (targeted), if a specific set of sample(s) is targeted by the attack, or (ii) *sample-generic* (indiscriminate), if any sample can be affected.

`Error Specificity.` It determines how the attack influences the model's predictions. It can be: (i) *error-specific*, if the attacker aims to have a sample misclassified as a specific class; or (ii) *error-generic*, if the attacker attempts to have a sample misclassified as any class different from the true class.

Attacker's Knowledge. The attacker may get to know some details about the target system, including information about: (i) the (clean) training data $\mathcal{D}$, (ii) the model being used $\mathcal{M}$, and (iii) the test data $\mathcal{T}$. The first component considers how much knowledge the attacker has on the training data. The second component refers to the ability of the attacker to access

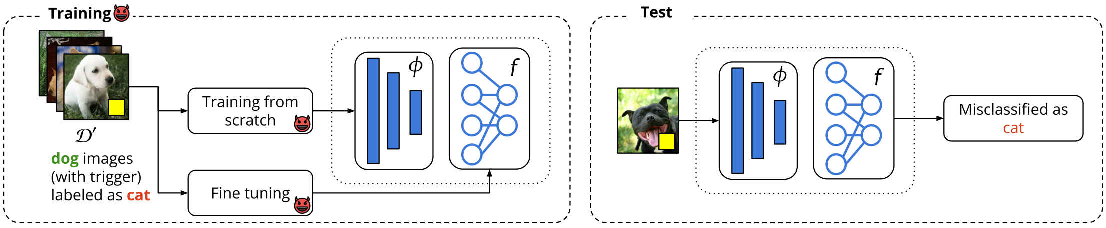


Fig. 3.1: *Training (left) and test (right) pipeline. The victim collects a training dataset $\mathcal{D}'$ from an untrusted source. The training or fine tuning algorithm uses these data to train a model $\mathcal{M}$, composed of a feature extractor ϕ, and a classification layer f. In the case of fine tuning, only f is modified, while the feature representation ϕ is left untouched. At test time, some test samples may be manipulated by the attacker to exploit the poisoned model and induce misclassification errors.*

the target model, including its internal (trained) parameters, but also additional information like hyperparameters, initialization, and the training algorithm. The third component specifies if the attacker knows in advance (or has access to) the samples that should be misclassified at test time. Although not explicitly mentioned in previous work, we have found that the knowledge of test samples is crucial for some attacks to work as expected. Clearly, attacks that are designed to work on specific test instances are not expected to generalize to different test samples (e.g., to other samples belonging to the same class). Depending on the combination of the previously-defined properties, we can define two main attack settings, as detailed below.

`White-Box Attacks.` The attacker has complete knowledge about the targeted system. Although not always representative of practical cases, this setting allows us to perform a worst-case analysis, and it is particularly helpful for evaluating defenses.

`Black-Box Attacks.` Black-box attacks can be subdivided into two main categories: black-box *transfer* attacks, and black-box *query* attacks. Although generally referred to as a black-box attack, *black-box transfer attacks* assume that the attacker has partial knowledge of the training data and/or the target model. In particular, the attacker is assumed to be able to collect a surrogate dataset and use it to train a surrogate model approximating the target. Then, white-box attacks can be computed against the surrogate model, and subsequently *transferred* against the target model. Under some mild conditions, such attacks have been shown to transfer successfully to the target model with high probability [73]. It is also worth remarking that *black-box query attacks* can also be staged against a target model, by only sending input queries to the model and observing its predictions to iteratively refine the attack, without exploiting any additional knowledge [47; 64; 211; 268]. However, to date, most of the poisoning attacks staged against learning algorithms in black-box settings exploit surrogate models and attack transferability.

Attacker's Capability. The attacker's capability is defined in terms of how the attacker can influence the *learning setting*, and on the *data perturbation* that can be applied to training and/or test samples.

`Influence on Learning Setting.` The three learning settings described in Sec. 3.2.1 open the door towards different data poisoning attacks. In both *training-from-scratch* (TS) and *fine-tuning* (FT) scenarios, the attacker alters a subset of the training dataset collected and used by the victim to train or fine-tune the machine learning model. Conversely, in the *model-training* (MT) scenario, as firstly hypothesized by Gu et al. [108], the attacker acts as a malicious third-party trainer, or as a man-in-the-middle, controlling the training process. The attacker tampers with the training procedure and returns to the victim user a model that behaves according to their goal. The advantage for the attacker is the victim will never be aware of the training dataset actually used. However, to keep their attack stealthy, the attacker must ensure that the provided model

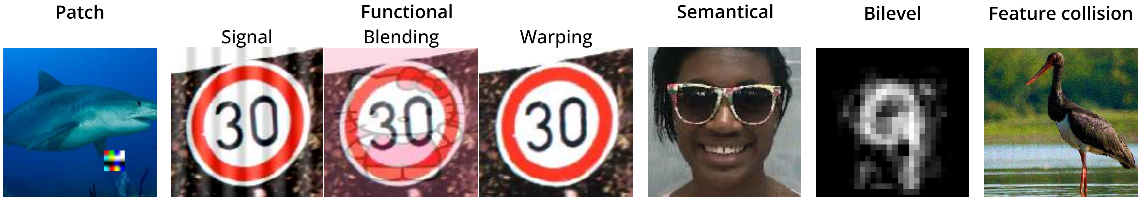


Fig. 3.2: *Visual examples of data perturbation noise. The first four figures show some examples of patch, functional, and semantical triggers. For functional triggers we consider signal [15], blending [50], and warping [205] transformations. The remaining two depict poisoning samples crafted with a bilevel attack with visible noise, and a clean-label feature collision attack with imperceptible noise.*

retains high prediction accuracy, making sure to pass the validation phase without suspicion from the victim user. The attacker's knowledge, discussed in Sec. 3.2.2, is defined depending on the setting under consideration. In the *model-training* and *training-from-scratch* settings, $\mathcal{D}'$ and $\mathcal{M}$ refer to the training data and algorithm used for training the model from random initialization of its weights. Conversely, in the *fine-tuning* setting, $\mathcal{D}'$ and $\mathcal{M}$ refer to the fine-tuning dataset and learning algorithm, respectively.

`Data Perturbation.` Staging a poisoning attack requires the attacker to manipulate a given fraction (p) of the training data. In some cases, i.e., in backdoor attacks, the attacker is also required to manipulate the test samples that are under their control, by adding an appropriate trigger to activate the previously-implanted backdoor at test time. More specifically, poisoning attacks can alter a fraction of the training labels and/or apply a (different) perturbation to each of the training (poisoning) samples. If the attack only modifies the training labels, but it does not perturb any training sample, it is often referred to as a *label-flip* poisoning attack. Conversely, if the training labels are not modified (e.g., if they are validated or assigned by human experts or automated labeling procedures), the attacker can stage a so-called *clean-label* poisoning attack. Such attacks only slightly modify the poisoning samples, using imperceptible perturbations that preserve the original semantics of the input samples along with their class labels [238]. We define the strategies used to manipulate training and test data in poisoning attacks in the next section.

Attack Strategy. The attack strategy amounts to defining how the attacker manipulates data to stage the desired poisoning attack. Both indiscriminate and targeted poisoning attacks only alter the training data, while backdoor attacks also require embedding the trigger within the test samples to be misclassified. We revise the corresponding data manipulation strategies in the following.

`Training Data Perturbation (`δ`).` Two main categories of perturbation have been used to mount poisoning attacks. The former includes perturbations which are found by solving an optimization problem, either formalized as a *bilevel* (BL) programming problem, or as a *feature-collision* (FC) problem. The latter involves the manipulation of training samples in targeted and backdoor poisoning attacks such that they collide with the target samples in the given representation space, to induce misclassification of such target samples in an attacker-chosen class. When it comes to backdoor attacks, three main types of triggers exist, which can be applied to training samples to implant the backdoor during learning: *patch triggers* (T^P), which consist of replacing a small subset of contiguous input features with a patch pattern in the input sample; *functional triggers* (T^F), which are embedded into the input sample via a blending function; and *semantical triggers* (T^S), which perturb the given input while preserving its semantics (e.g., modifying face images by adding sunglasses, or altering the face expression, but preserving the user identity). The choice of this strategy plays a fundamental role since it influences the computational effort, effectiveness,

and stealthiness of the attack. More concretely, the trigger strategies are less computationally demanding, as they do not require optimizing the perturbation, but the attack may be less effective and easier to detect. Conversely, an optimized approach can enhance the effectiveness and stealthiness of the attack, at the cost of being more computationally demanding.
In Fig. 3.2 we give some examples of patch, functional, and semantical triggers, one example of a poisoning attack optimized with bilevel programming, and one example of a *clean-label* feature-collision attack.

`Test Data Perturbation` ($\boldsymbol{t}$). During operation, i.e., at test time, the attacker can submit malicious samples to exploit potential vulnerabilities that were previously implanted during model training, via a backdoor attack. More concretely backdoor attacks are activated when a specific trigger $\boldsymbol{t}$ is present in the test samples. Normally, the test-time trigger is required to exactly match the trigger implanted during training, thus including patch, functional, and semantical triggers. Further investigation on backdoor poisoning are given in Sec. 3.3.3.

3.2.3 Defense Framework

In this section, we introduce the main strategies that can be used to counter poisoning attacks, based on different assumptions made on the defender's goal, knowledge and capability.

Defender's Goal. The defender's goal is to preserve the integrity, availability, and privacy of their ML model, i.e., to mitigate any kind of security violation that might be caused by an attack. The defender thus adopts appropriate countermeasures to alleviate the effect of possible attacks, without significantly affecting the behavior of the model for legitimate users.

Defender's Knowledge and Capability. The defender's knowledge and capability determine in which learning setting a defense can be applied. We identify four aspects that influence how the defender can operate to protect the model: (i) having access to the (poisoned) training data $\mathcal{D}'$, and to (ii) a separate, clean validation set $\mathcal{V}$, and (iii) having control on the training procedure $\mathcal{W}$, and on (iv) the model's parameters $\boldsymbol{\theta}$. We will see in more detail how these assumptions are matched to each defense in Sec. 3.4.

Defense Strategy. The defense strategy defines how the defender operates to protect the system from malicious attacks before deployment (i.e., at training time), and after the model's deployment (i.e., at test time). We identify six distinct categories of defenses:

1. *training data sanitization*, which aims to remove potentially-harmful training points before training the model;
2. *robust training*, which alters the training procedure to limit the influence of malicious points;
3. *model inspection*, which returns for a given model whether it has been compromised (e.g., by a backdoor attack);
4. *model sanitization*, which cleans the model to remove potential backdoors or targeted poisoning attempts;
5. *trigger reconstruction*, which recovers the trigger embedded in a backdoored network;
6. *test data sanitization*, which filters potentially-triggered samples presented at test time.

These defenses essentially work by either (i) cleaning the data or (ii) modifying the model. In the former case, the defender aims to sanitize training or test data. *Training data sanitization* and *test data sanitization* as thus two strategies adopted respectively at training and at test time to mitigate the influence of data poisoning attacks. Alternatively, the defender can act directly on the model, by (i) identifying possible internal vulnerabilities and removing/fixing components that lead to

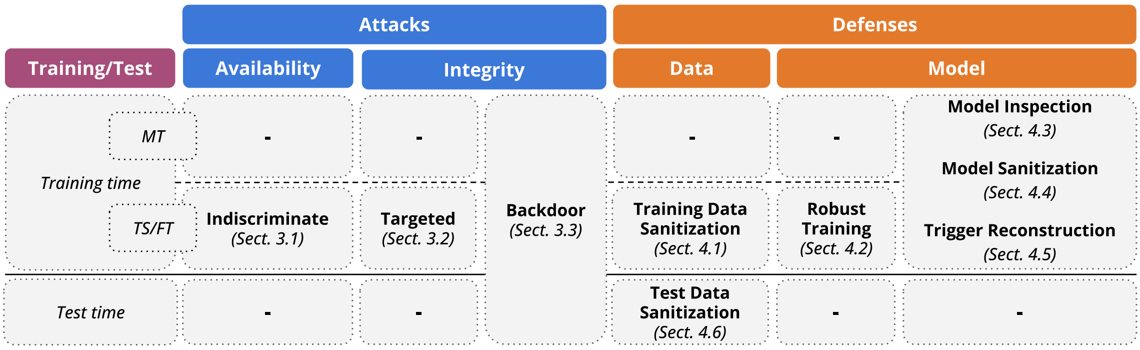

Attacks
Defenses
Training/Test
Availability
Integrity
Data
Model
MT
Training time
TS/FT
Test time
Indiscriminate (Sect. 3.1)
Targeted (Sect. 3.2)
Backdoor (Sect. 3.3)
Training Data Sanitization (Sect. 4.1)
Robust Training (Sect. 4.2)
Model Inspection (Sect. 4.3)
Model Sanitization (Sect. 4.4)
Trigger Reconstruction (Sect. 4.5)
Test Data Sanitization (Sect. 4.6)

Fig. 3.3: *Conceptual overview of poisoning attacks and defenses according to our framework. Attacks are categorized based on whether they compromise system integrity or availability. Defenses are categorized based on whether they sanitize data or modify the learning algorithm/model. Training-time (test-time) defenses are applied before (after) model deployment. Training-time interventions are also divided according to whether* model-training *(MT) is outsourced, or* training-from-scratch *(TS) /* fine-tuning *(FT) is performed.*

anomalous behavior/classifications, or by (ii) changing the training procedure to make the model less susceptible to training data manipulations. The first approach is employed in *model inspection*, *trigger reconstruction* and *model sanitization* defensive mechanisms. The second approach, instead, includes algorithms that operate at the training level to implement *robust training* mechanisms.

3.2.4 Poisoning Attacks and Defenses

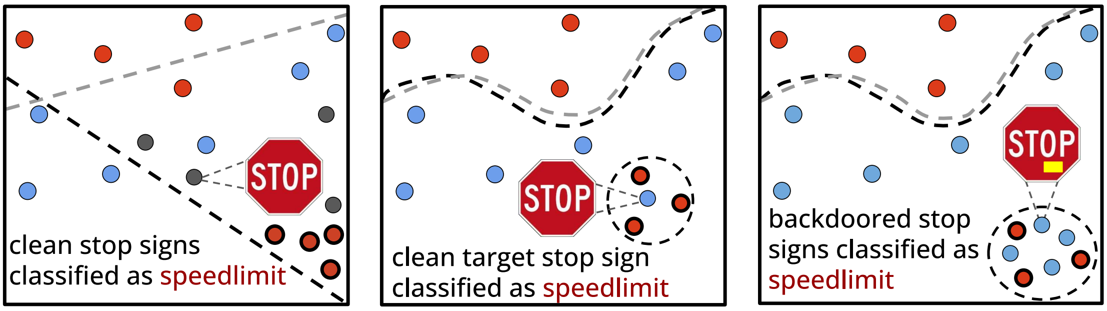
We provide in Fig. 3.3 a preliminary, high-level categorization of attacks and defenses according to our framework. This simplified taxonomy categorizes attacks and defenses based on whether they are applied at training time (and in which learning setting) or at test time; whether the attack aims to violate integrity or availability;[1] and whether the defense aims to sanitize data or modify the learning algorithm/model. As one may note, indiscriminate and targeted poisoning only manipulate data at training time to violate availability and integrity, respectively, and they are typically staged in the *training-from-scratch* (TS) or *fine-tuning* (FT) learning settings. Backdoor attacks, in addition, require manipulating the test data to embed the trigger and cause the desired misclassifications, with the goal of violating integrity. Such attacks can be ideally staged in any of the considered learning settings. For defenses, data sanitization strategies can be applied either at training time or at test time, while defenses that modify the learning algorithm or aim to sanitize the model can be applied clearly only at training time (i.e., before model deployment). To conclude, while being simplified, we do believe that this conceptual overview of attacks and defenses provides a comprehensive understanding of the main assumptions behind each poisoning attack and defense strategy. Accordingly, we are now ready to delve into a more detailed description of attacks and defenses in Sects. 3.3 and 3.4.

3.3 Poisoning Attacks

We now take advantage of the previous framework to give an overview of the existing attacks according to the corresponding violation and strategy. Our survey is mostly focused on poisoning classification models for computer vision, which encompasses most of the work related to poisoning attacks and defenses. A compact summary of all attacks from the vision domain is given in Table 3.1.

---

[1] To our knowledge, no poisoning attack violating a model's privacy has been considered so far, so we omit the privacy dimension from this representation.



(a) *Indiscriminate attack.* **(b)** *Targeted attack.* **(c)** *Backdoor attack.*

Fig. 3.4: *Conceptual representation of the impact of indiscriminate, targeted, and backdoor poisoning on the learned decision function. We depict the feature representations of the speed limit sign (red dots) and stop signs (blue dots). The poisoning samples (solid black border) change the original decision boundary (dashed gray) to a poisoned variant (dashed black).*

3.3.1 Indiscriminate (Availability) Poisoning

Indiscriminate poisoning attacks represent the first class of poisoning attacks against ML algorithms. The attacker aims to subvert the system functionalities, compromising its availability for legitimate users by poisoning the training data. More concretely, the attacker aims to cause misclassification on clean validation samples by injecting new malicious samples or perturbing existing ones in the training dataset. In Fig. 3.4a we consider the case where an attacker poisons a linear street-sign classifier to have stop signs misclassified as speed limits. The adversary injects poisoning samples to rotate the classifier's decision boundary, thus compromising the victim's model performance. In the following, we present the strategies developed in existing works and categorize them in Table 3.1. Although they could also operate on the *fine-tuning* (FT) scenario, existing works have been proposed only in the *training-from-scratch* (TS) setting. By contrast, their application in the *model-training* (MT) scenario would not be feasible, as the model, with reduced accuracy due to the attack, would not pass the user validation phase. To be appliable in the latter scenario without detection, indiscriminate attacks must compromise the system's availability but not increase the classification error. This has been recently done by Cinà et al. [61], who proposed a so-called *sponge* poisoning attack aimed to increase the model's prediction latency.

Label-Flip Poisoning The most straightforward strategy to stage poisoning attacks against ML is label-flip, originally proposed by Biggio et al. [26]. The adversary does not perturb the feature values, but they mislabel a subset of samples in the training dataset, compromising the performance accuracy of ML models such as Support Vector Machines (SVMs). Beyond that, Xiao et al. [292] showed that random flips could have far-from-optimal performance, which would require solving an NP-hard optimization problem. Due to its intractability, heuristic strategies have been proposed by Xiao et al. [292], and later by Xiao et al. [294], to efficiently approximate the optimal formulation.

Bilevel Poisoning In this case, the attacker manipulates the training samples and their labels. The pioneering work in this direction was proposed by Biggio et al. [27], where a gradient-based indiscriminate poisoning attack is exploited against SVMs. They exploited *implicit differentiation* to derive the gradient required to optimize the poisoning samples by their iterative algorithm. Until convergence, the poisoning samples are iteratively updated following the implicit gradient, directing towards maximization of the model's validation error. Mathematically speaking, this idea

Table 3.1: *Taxonomy of existing poisoning attacks, according to the attack framework defined in Sect. 3.2. The presence of the* ✔ *indicates that the corresponding properties is satisfied by the attack. For the attacker's knowledge we use:* ○ *when the attacker has knowledge of the corresponding component;* ◐ *if the attacker uses a surrogate to mount the attack;* ● *if the attacker does not require that knowledge. In the attacker's capabilities we use MT, TS and FT as acronyms for* model-training, training-from-scratch, *and* fine-tuning *learning settings.* ◔, ◑, ⬤ *represent the amount of poisoning: small* ($\leq$ 10%)*, medium* ($\leq$ 30%)*, or high percentage of the training set. The columns* $\boldsymbol{\delta}$ *and* $\boldsymbol{t}$ *define the training and test strategies: optimized bilevel – BL, feature collision – FC and trigger – T.*

Section		Attacks	Goal		Knowledge			Capability			Strategy		Model
			Sample Specific	Error Specific	$\mathcal{D}$	$\mathcal{M}$	$\mathcal{T}$	Setting	Clean Label	p	$\boldsymbol{\delta}$	$\boldsymbol{t}$	DNN
Indiscriminate	3.3.1	Biggio et al. [26], Xiao et al. [292],			○	○		TS		◑	LF	-	
		Xiao et al. [294], Paudice et al. [214]			○	○		TS		◑			
	3.3.1	Biggio et al. [27]; Xiao et al. [293] Frederickson et al. [96]			○	○		TS		◑	BL	-	
		BetaPoison [60], Ma et al. [182]		✔	○	◐		TS		◑			
		Solans et al. [248]; Demontis et al. [73]			◐	◐		TS		◑			
		Muñoz-González et al. [197]; Yang et al. [301]			◐	◐		TS		◑			✔
	3.3.1	Mei and Zhu [191]		✔	○	○		TS	✔	⬤	BL	-	
		Feng et al. [93]		✔	◐	◐		TS	✔	⬤			✔
		Fowl et al. [94]		✔	◐	◐		TS	✔	⬤			✔
Targeted	3.3.2	Koh and Liang [144]	✔	✔	○	○	✔	FT		◔	BL	-	
		Muñoz-González et al. [197]	✔	✔	◐	◐		TS		◔			✔
		Jagielski et al. [134]		✔	◐	◐		TS/FT		◔			✔
	3.3.2	PoisonFrog [238]	✔	✔	○	○	✔	FT	✔	◔	FC	-	✔
		Guo and Liu [109], StingRay[256] ConvexPolytope [323], BullseyePolytope [2]	✔	✔	◐	◐	✔	FT	✔	◔			✔
	3.3.2	Geiping et al. [101], MetaPoison [129]	✔	✔	◐	◐	✔	TS	✔	◔	BL	-	✔
Backdoor	3.3.3	BadNet [108], LatentBackdoor [303]	✔	✔	○	○		MT		◔	T^P	T^P	✔
		BaN [230]	✔	✔	○	○		MT		◑			✔
		TrojanNN [173]	✔	✔	●	○		MT		◔			✔
	3.3.3	WaNET [205], Li et al. [160], DFST [54]	✔	✔	○	○		MT		◑	T^F	T^F	✔
		Refool [175]	✔	✔	○	○		TS	✔	◑			✔
		SIG [15]	✔	✔	●	●		TS	✔	◑			✔
		Chen et al. [50]; Zhong et al. [322]	✔	✔	●	●		TS/FT		◔			✔
	3.3.3	FaceHack [234]	✔	✔	○	○		MT		◑	T^S	T^S	✔
		Chen et al. [50]	✔	✔	●	●		TS/FT		◔			✔
		Wenger et al. [282]	✔	✔	○	●		FT		◑			✔
	3.3.3	Nguyen and Tran [204], LIRA [76]	✔	✔	○	○		MT		◑	BL	T^F	✔
		Li et al. [160]	✔	✔	●	○		MT		◔			✔
		Li et al. [165]	✔	✔	○	◐		TS		◑			✔
		Zhong et al. [322]	✔	✔	◐	◐		TS/FT		◔			✔
	3.3.3	HiddenTrigger [229]	✔	✔	○	○		FT	✔	◑	FC	T^P	✔
		Turner et al. [270]	✔	✔	◐	◐		TS	✔	◔			✔
	3.3.3	Souri et al. [251]	✔	✔	◐	◐		TS	✔	◔	BL	T^P	✔

corresponds to treating the poisoning task as a bilevel optimization problem:

$$\max_{\boldsymbol{\delta} \in \Delta} \quad L(\mathcal{V}, \mathcal{M}, \boldsymbol{\theta}^\star)\,, \tag{3.1}$$

$$\text{s.t.} \quad \boldsymbol{\theta}^\star \in \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D} \cup \mathcal{D}_p^{\boldsymbol{\delta}}, \mathcal{M}, \boldsymbol{\theta})\,. \tag{3.2}$$

with Δ being the set of admissible manipulation of the training samples that preserve the constraints imposed by the attackers (e.g., ℓ_p, or box-constraints)[2]. We define with $\mathcal{D}_p = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ the training data controlled by the attacker, before any perturbation is applied, being y_i the pristine label of sample $\boldsymbol{x}_i$ and n the number of samples in $\mathcal{D}_p$. We then denote with $\mathcal{D}_p^{\boldsymbol{\delta}}$ the corresponding poisoning dataset manipulated according to the perturbation parameter $\boldsymbol{\delta}$. The attacker optimizes the perturbation $\boldsymbol{\delta}$ (applied to the poisoning samples $\mathcal{D}_p$) to increase the error/loss L of the target model $\mathcal{M}$ on the clean validation samples $\mathcal{V}$. Our formulation in Eqs. (3.1)-(3.2) encompass both dirty or clean-label attacks according to the nature of $\mathcal{D}_p^{\boldsymbol{\delta}}$. For example, we can define $\mathcal{D}_p^{\boldsymbol{\delta}} = \{(\boldsymbol{x}_i + \boldsymbol{\delta}_i, y_i')\}_{i=1}^n$[3], being y_i' the poisoning label chosen by the attacker, with $y_i' = y_i$ for a clean-label attack and $y_i' \neq y_i$ for a dirty-label attack.

Solving this bilevel optimization is challenging, since the inner and the outer problems in Eqs. (3.1)-(3.2) have conflicting objectives. More concretely, the inner objective is a regularized empirical risk minimization, while the outer one is empirical risk maximization, both considering data from the same distribution.

A similar approach was later generalized in Xiao et al. [293] and Frederickson et al. [96] to target feature selection algorithms (i.e., LASSO, ridge regression, and elastic net).

Subsequent work tried to analyze the robustness of ML models when the attacker has limited knowledge about the training dataset or the victim's classifier. In this scenario, the most investigated methodology is given by the *transferability* of the attack [73; 182; 248]. The attacker crafts the poisoning samples using surrogate datasets and/or models and then transfers the attack to another target model. This approach has proven effective for corrupting logistic classifiers [73], algorithmic fairness [248], and differentially-private learners [182]. More details about the transferability of poisoning attacks are presented in Sec. 3.3.6.

Differently from previous work, Cinà et al. [60] observed that a simple heuristic strategy and a variable reduction technique can reach noticeable results against linear classifiers with increased computational efficiency. More concretely, the authors showed how previous gradient-based approaches could be affected by several factors (e.g., loss landscape) that degrade their performance in terms of computation time and attack efficiency.

Although effective, the aforementioned poisoning attacks have been designed to fool models with a relatively small number of parameters. More recently, Muñoz-González et al. [197] showed that devising poisoning attacks against larger models, such as convolutional neural networks, can be computationally and memory demanding. To this end, Muñoz-González et al. [197] pioneered the idea of adapting hyperparameter optimization methods, which aim to solve bilevel programming problems more efficiently, in the context of poisoning attacks. The authors proposed a back-gradient descent technique to optimize poisoning samples, drastically reducing the attack complexity. The underlying idea is to back-propagate the gradient of the objective function to the poisoning samples while learning the poisoned model. However, they assume the objective function is sufficiently smooth to trace the gradient backward correctly. Another way explored in Yang et al. [301] was to train a generative model from which the poisoning samples are generated, thus increasing the generation rate.

Bilevel Poisoning (Clean-Label) Previous work examined in Sec. 3.3.1 assumes that the attacker has access to a small percentage of the training data and can alter both features and labels. Similar attacks have been staged by assuming that the attacker can control an extensively larger

---

[2] For example, the attacker can constraint the perturbation magnitude of $\boldsymbol{\delta}$ imposing $\|\boldsymbol{\delta}\|_p \leq \epsilon$ with $\Delta = \{\boldsymbol{\delta} \in \mathbb{R}^{n \times d} \mid \|\boldsymbol{\delta}\|_p \leq \epsilon\}$.

[3] In this example we used $\boldsymbol{\delta}$ as additive noise. To be more generic we can define a manipulation function h parametrized by $\boldsymbol{\delta}$ and the sample $\boldsymbol{x}$ to perturb.

fraction of the training set while only slightly manipulating each poisoning sample to preserve its class label, i.e., performing a clean-label attack. This idea was introduced by Mei and Zhu [191], who considered manipulating the whole training set to arbitrarily define the importance of individual features on the predictions of convex learners. More recently, DeepConfuse [93] and Fowl et al. [94] proposed novel techniques to mount clean-label poisoning attacks against DNNs. In [93], the attacker trains a generative model, similarly to [301], to craft clean-label poisoning samples which can compromise the victim's model. Inspired by recent developments proposed in [101], Fowl et al. [94] used a gradient alignment optimization technique to alter the training data imperceptibly but diminish the model's performance. Even though Feng et al. [93] and Fowl et al. [94] can target DNNs, the attacker is assumed to perturb a high fraction of samples in the training set. We do believe that this is a very demanding setting for poisoning attacks. In fact, such attacks are often possible because ML is trained on data collected in the wild (e.g., labeled through tools such as a mechanical Turk) or crowdsourced from multiple users; thus, it would be challenging for attackers in many applications to control a substantial fraction of these training data realistically. In conclusion, the quest for scalable, effective, and practical indiscriminate poisoning attacks on DNNs is still open. Accordingly, it remains unclear whether such attacks in practical settings can significantly subvert DNNs.

3.3.2 Targeted (Integrity) Poisoning

In contrast to indiscriminate poisoning, targeted poisoning attacks preserve the system's availability, functionality, and behavior for legitimate users, while causing misclassification of some specific target samples. Like indiscriminate poisoning, targeted poisoning attacks manipulate the training data but do not require modifying the test data.

An example of a targeted attack is given in Fig. 3.4b, where the classifier's decision function for clean samples is not significantly changed after poisoning, preserving the model's accuracy. However, the model isolated the target stop sign (grey) to be misclassified as a speed-limit sign. The system can still correctly classify most clean samples but outputs wrong predictions for the target stop sign.

In the following sections, we describe the targeted poisoning attacks categorized in Table 3.1. Notably, such attacks have been investigated both in the *training-from-scratch* (TS) and *fine-tuning* (FT) settings, defined in Sec. 3.2.1.

Bilevel Poisoning In Sec. 3.3.1, we reviewed the work in Muñoz-González et al. [197]. In addition to indiscriminate poisoning, the authors also formulated targeted poisoning attacks as:

$$\min_{\boldsymbol{\delta} \in \Delta} \quad L(\mathcal{V}, \mathcal{M}, \boldsymbol{\theta}^\star) + L(\mathcal{V}_t, \mathcal{M}, \boldsymbol{\theta}^\star)\,, \tag{3.3}$$

$$\text{s.t.} \quad \boldsymbol{\theta}^\star \in \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D} \cup \mathcal{D}_p^{\boldsymbol{\delta}}, \mathcal{M}, \boldsymbol{\theta})\,. \tag{3.4}$$

Within this formulation, the attacker optimizes the perturbation $\boldsymbol{\delta}$ on the poisoning samples $\mathcal{D}_p$ to have a set of target (validation) samples $\mathcal{V}_t$ misclassified while preserving the accuracy on the clean (validation) samples in $\mathcal{V}$. It is worth noting that the attack is optimized on a set of validation samples, then evaluated on a separate set of test samples. The underlying rationale is that the attacker can not typically control the specific realization of the target instances at test time (e.g., if images are acquired from a camera sensor, the environmental and acquisition conditions can not be controlled), and the attack is thus expected to generalize correctly to that case.

A similar attack was introduced by Koh and Liang [144], to show the equivalence between gradient-based (bilevel) poisoning attacks and influence functions, i.e., functions defined in the area of robust statistics that identify the most relevant training points influencing specific predictions. Notably, these authors were the first to consider the *fine-tuning* (FT) scenario in their experiments, training the classification function f (i.e., an SVM with the RBF kernel) on top of a feature representation ϕ extracted from an internal layer of a DNN. Although these two bilevel optimization strategies have been proven effective, they remain too computationally demanding to be applied to DNNs.

Jagielski et al. [134] showed how to generalize targeted poisoning attacks to an entire subpopulation in the data distribution while reducing the computational cost. To create subpopulations, the attacker selects data samples by matching their features or clustering them in feature space. The poisoning attack can be performed either by label flipping or linearizing the influence function to approximate the poisoning gradients, thus reducing the computational cost of the attack. Muñoz-González et al. [197] and Jagielski et al. [134] define a more ambitious goal for the attack compared to Koh and Liang [144], as their attacks aim to generalize to all samples coming from the target distribution or the given subpopulation. Specifically, the attack by Koh and Liang [144] is tailored for misleading the model only for some specific test samples, which means considering the test set $\mathcal{T}$ rather than a validation set $\mathcal{V}_t$ in Eq. (3.3). However, the cost of the attack by Muñoz-González et al. [197] is relatively high due to the need to solve a bilevel problem, while the attack by Jagielski et al. [134] is faster, but it does not achieve the same success rate on all subpopulations.

Feature Collision (Clean-Label) This category of attacks is based on a heuristic strategy named *feature collision*, suited to the so-called *fine-tuning* (FT) scenario, which avoids the need to solve a complex bilevel problem to optimize poisoning attacks. In particular, PoisonFrog [238] was the first work proposing this idea, which can be formalized as:[4]

$$\min_{\boldsymbol{\delta}} \quad \|\phi(\boldsymbol{x}+\boldsymbol{\delta}) - \phi(\boldsymbol{z})\|_2^2 \,. \tag{3.5}$$

This attack amounts to creating a poisoning sample $\boldsymbol{x}+\boldsymbol{\delta}$ that collides with the target test sample $\boldsymbol{z} \in \mathcal{T}$ in the feature space so that the fine-tuned model predicts $\boldsymbol{z}$ according to the poisoning label associated with $\boldsymbol{x}$. To this end, the adversary leverages the feature extractor ϕ to minimize the distance of the poisoning sample from the target in the feature space. Moreover, the authors observed that, due to the complexity and nonlinear behavior of ϕ, the poisoning samples only need to be slightly perturbed. Even samples from different distributions can be slightly perturbed to match the feature representation of target samples from other classes. Similarly, Guo and Liu [109] adopted *feature collision* to stage the attack, but they extended the attack's objective function to further increase the poisoning effectiveness.

Nevertheless, although this strategy turns out to be effective, it assumes that the feature extractor is fixed and that it is not updated during the fine-tuning process. Moreover, StringRay [256], ConvexPolytope [323], and BullseyePolytope [2] observed that when reducing the attacker's knowledge, the poisoning effectiveness decreases. These works showed that *feature collision* is not practical if the attacker does not know exactly the details of the feature extractor, as the embedding of poisoning samples may not be consistent across different feature extractors. To mitigate these difficulties, ConvexPolytope [323] and BullseyePolytope [2] optimize the poisoning samples against *ensemble models*, constructing a convex polytope around the target samples to enhance the effectiveness of the attack. The underlying idea is that constructing poisoning points against ensemble models may improve attack transferability. The authors further optimize the poisoning samples by establishing a strong connection among all the layers and the embeddings of the poisoning samples, partially overcoming the assumption that the feature extractor ϕ remains fixed.

All these approaches create clean-label samples, as first proposed in Shafahi et al. [238], to stay undetected even when humans validate the class labels of training points. This is possible as these attacks are staged against deep models since, for these models, small (adversarial) perturbations of samples in the input space correspond to large changes in their feature representations.

Bilevel Poisoning (Clean-Label) Although *feature collision* attacks are effective, they may not result in optimal accuracy, and they do not minimize the number of poisoned points to change the model's prediction on a single test point. Moreover, they assume that the training process is not significantly changing the feature embedding. Indeed, when the whole model is trained from scratch, these strategies may not work properly as poisoning samples can be embedded differently.

---

[4] We neglect the penalty term used to increase the attack's stealthiness introduced in [238] as it is not related to the *feature collision* strategy.

Recent developments, including MetaPoison [129] and the work by Geiping et al. [101], tackle the targeted poisoning attack in the *training-from-scratch* (TS) scenario, while ensuring the clean-label property. These approaches are derived from the bilevel formulation in Eqs. (3.3)-(3.4), but they exploit distinct and more scalable approaches to target DNNs, and optimize the attack directly against the test samples $\mathcal{T}$ as done in [144]. More concretely, MetaPoison [129] uses a meta-learning algorithm, as done by Muñoz-González et al. [197], to decrease the computational complexity of the attack. They further enhance the transferability of their attack by optimizing the poisoning samples against an ensemble of neural networks, trained with different hyperparameter configurations and algorithms (e.g., weight initialization, number of epochs). Geiping et al. [101] craft poisoning samples to maximize the alignment between the inner loss and the outer loss in Eqs. (3.3)-(3.4). The authors observed that matching the gradient direction of adversarial examples is an effective strategy for attacking DNNs trained from scratch, even on large training datasets. Although modern *feature collision* or optimized strategies are emerging with notable results for targeted attacks, their performance, especially in black-box settings, still demands further investigation.

3.3.3 Backdoor (Integrity) Poisoning

Backdoor poisoning attacks aim to cause an integrity violation. In particular, for any test sample containing a specific pattern, i.e., the so-called *backdoor trigger*, they aim to induce a misclassification without affecting the classification of clean test samples. The backdoor trigger is known only to the attacker, making it challenging for the defender to evaluate whether or not a given model provided to them has been backdoored during training. In Fig. 3.4c we consider the case where the attacker provides a backdoored street-sign detector that has good accuracy for classifying street signs in most circumstances. However, the classifier has successfully learned the backdoor data distribution and will output speed-limit predictions for any stop sign containing the backdoor trigger. In the following sections, we describe backdoor attacks following the categorization given in Table 3.1. Notably, such attacks have been initially staged in the *model-training* (MT) setting, assuming that the user outsources the training process to an untrusted third-party service, but they have then been extended also to the *training-from-scratch* (TS) and *fine-tuning* (FT) scenarios.

Trigger Poisoning Earlier work in backdoor attacks considered three main families of backdoor triggers, i.e., *patch*, *functional*, and *semantical* triggers, as discussed below.

`Patch.` The first threat vector of attack for backdoor poisoning has been investigated in BadNets [108]. The authors considered the case where the user outsources the training process of a DNN to a third-party service, which maliciously alters the training dataset to implant a backdoor in the model. To this end, the attacker picks a random subset of the training data, blends the backdoor trigger into them, and changes their corresponding labels according to an attacker-chosen class. A similar idea has been investigated further in LatentBackdoor [303] and TrojanNN [173], where the backdoor trigger is designed to maximize the response of selected internal neurons, thus reducing the training data needed to plant the trigger. Additionally, LatentBackdoor [303] designed the trigger to survive even if the last layers are fine-tuned with novel clean data, while TrojanNN [173] does not need access to the training data as a reverse-engineering procedure is applied to create a surrogate dataset. All these attacks assume that the trigger is always placed in the same position, limiting their application against specific defense strategies [250; 12; 44]. To overcome this issue, BaN [230] introduced different backdoor attacks where the trigger can be attached in various locations of the input image. The underlying idea was to force the model to learn the backdoor trigger and make it location invariant.

`Functional.` The patch strategy is based on the idea that poisoning samples repeatedly present a fixed pattern as a trigger, which, however, may be detected upon human validation of training samples (in the TS and FT scenarios, at least). In contrast, a functional trigger represents a stealthier strategy as the corresponding trigger perturbation is slightly spaced throughout the image or changes according to the input. Some works assume to slightly perturb the entire image so that those small variations are not detectable by humans but evident enough to mislead the model. In WaNET [205] warping functions are used to generate invisible backdoor triggers (see Fig. 3.2). Moreover, the authors enforced the model to distinguish the backdoor warping functions from other pristine ones. In Li et al. [160] *steganography* algorithms are used to hide the trigger into the training data. Specifically, the attacker replaces the least significant bits to contain the trigger's binary string. In DFST [54] style transfer generative models are exploited to generate and blend the trigger. However, the aforementioned poisoning approaches assume that the attacker can change the labeling process and that no human inspection is done of the training data. This assumption is then relaxed by Barni et al. [15] and Liu et al. [175], where clean-label backdoor poisoning attacks are considered; in particular, Liu et al. [175] used natural reflection effects as trigger to backdoor the system, while Barni et al. [15] used an invisible sinusoidal signal as backdoor trigger (see Fig. 3.2). More practical scenarios, where the attacker is assumed to have limited knowledge, have been investigated by Chen et al. [50] and Zhong et al. [322]. In these two works, the authors used the idea of blending fixed patterns to backdoor the model. In the former approach, Chen et al. [50] assumes that the attacker blends image patterns into the training data and tunes the blend ratio to create almost invisible triggers while impacting the backdoor's effectiveness. In the latter, Zhong et al. [322] assumes that an invisible grid pattern is generated to increase the pixel's intensity, and its effectiveness is tested in the TS and FT settings.

`Semantical.` The semantical strategy incorporates the idea that backdoor triggers should be feasible and stealthy. For example, Sarkar et al. [234] used facial expressions or image filters (e.g., old-age, smile) as backdoor triggers against real-world facial recognition systems. At training time, the backdoor trigger is injected into the training data to cause the model to associate a smile filter with the authorization of a user. At test time, the attacker can use the same filter to mislead classification. Similarly, Chen et al. [50] and Wenger et al. [282] tried to poison face-recognition systems by blending physically-implementable objects (e.g., sunglasses, earrings) as triggers.

Bilevel Poisoning Trigger-based strategies assume that the attacker uses a predefined perturbation to mount the attack. However, an alternative strategy for the attacker is to learn the trigger/perturbation itself to enhance the backdoor effectiveness. To this end, even backdoor poisoning can be formalized as a bilevel optimization problem:

$$\min_{\boldsymbol{\delta}\in\Delta} \quad L(\mathcal{V},\mathcal{M},\boldsymbol{\theta}^\star) + L(\mathcal{V}_t^{\boldsymbol{t}},\mathcal{M},\boldsymbol{\theta}^\star)\,, \tag{3.6}$$

$$\text{s.t.} \quad \boldsymbol{\theta}^\star \in \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D}\cup\mathcal{D}_p^{\boldsymbol{\delta}},\mathcal{M},\boldsymbol{\theta})\,. \tag{3.7}$$

Here, the attacker optimizes the training perturbation $\boldsymbol{\delta}$ for poisoning samples in $\mathcal{D}_p$ to mislead the model's prediction for validation samples $\mathcal{V}_t$ containing the backdoor trigger $\boldsymbol{t}$. In contrast to indiscriminate and targeted attacks (in Sec. 3.3.1 and Sec. 3.3.2), the attacker injects the backdoor trigger in the validation samples $\boldsymbol{t}$ to cause misclassifications. Additionally, as for targeted poisoning, the error on $\mathcal{V}$ is minimized to preserve the system's functionality.

One way to address this bilevel formulation is to craft optimal poisoning samples using generative models [204; 76; 165], as also done in [301] for indiscriminate poisoning. Nguyen and Tran [204] trained the generative model with a loss that enforces the *diversity* and *noninterchangeable* of the trigger, while LIRA [76]'s generator is trained to enforce effectiveness and invisibility of the triggers. Conversely, Li et al. [165] used a generative neural network steganography technique to embed a backdoor string into poisoning samples. Another way is to perturb training samples with adversarial noise, as done by Li et al. [160] and Zhong et al. [322]. More concretely, in the former approach, the trigger maximizes the response of specific internal neurons, and a regularization term is introduced in the objective function to make the backdoor trigger invisible. In the latter work, the attacker looks for the minimum universal perturbation that pushes any input towards the decision boundary of a target class. The attacker can use this invisible perturbation trigger on any image, inducing the model to misclassify the target class.

Feature Collision (Clean-Label) The backdoor trigger visibility influences the stealthiness of the attack. A backdoor trigger that is too obvious can be easily spotted when the dataset is inspected [229]. However, Hidden Trigger [229] introduced the idea of using the *feature collision* strategy, seen in Sec. 3.3.2 and formulated in Eq. (3.5), to hide the trigger into natural target samples. The attacker first injects a random patch trigger into the training set, and then each poisoning sample is masked via *feature collision*. The resulting poisoning images are visually indistinguishable from the target and have a consistent label (i.e., they are clean-label), while the test samples with the patch trigger will collide with the poisoning samples in feature space ensuring that the attack works as expected.

Although the work in [229] implements an effective and stealthy clean-label attack, it is applicable only in the feature extractor ϕ is not updated. Such a limitation is mitigated by Turner et al. [270] who exploit a surrogate latent space, rather than ϕ, to interpolate the backdoor samples, hiding the training-time trigger. Moreover, the attacker can tune the trigger visibility at test time to enhance the attack's effectiveness.

Bilevel Poisoning (Clean-Label) Inspired by recent success of the gradient-alignment technique in [101] for targeted poisoning, Souri et al. [251] exploited the same bilevel-descending strategy to stage clean-label backdoor poisoning attacks in the *training-from-scratch* scenario. Similarly to Saha et al. [229] the training and the test data perturbations are different, enhancing the stealthiness of the attack and making it stronger against existing defenses.

3.3.4 Unifying Framework

Although the three poisoning attacks are detailed in Sects. 3.3.1-3.3.3 aim to cause different violations, they can be described by the following, generalized bilevel programming problem:

$$\max_{\boldsymbol{\delta}\in\Delta} \quad \alpha L(\mathcal{V}, \mathcal{M}, \boldsymbol{\theta}^\star) - \beta L(\mathcal{V}_t^{\boldsymbol{t}}, \mathcal{M}, \boldsymbol{\theta}^\star)\,, \tag{3.8}$$

$$\text{s.t.} \quad \boldsymbol{\theta}^\star \in \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D} \cup \mathcal{D}_p^{\boldsymbol{\delta}}, \mathcal{M}, \boldsymbol{\theta})\,, \tag{3.9}$$

The optimization program in Eqs. (3.8)-(3.9) aims to accomplish the attacker's goal, considering their capacity of tampering with the training set and knowledge of the victim model, by optimizing the perturbation $\boldsymbol{\delta}$ used to poison the training samples in $\mathcal{D}_p$. Additionally, as in Eqs. (3.1)-(3.7), the poisoning noise $\boldsymbol{\delta}$ belongs to Δ which encompass possible domain constraints or feature constraints to improve stealthiness of the attack (e.g., invisibility of the trigger). The test data perturbation $\boldsymbol{t}$ is absent (i.e., $\boldsymbol{t} = \boldsymbol{0}$), for indiscriminate and target poisoning. For backdoor poisoning, $\boldsymbol{t}$ is pre-defined/optimized by the attacker before training, unlike from adversarial examples [28; 105] where the perturbation $\boldsymbol{t}$ is optimized at test time. The coefficients α and β are calibrated according to the attacker's desired violation. We can set: (i) $\alpha = 1(-1)$ and $\beta = 0$ for error-generic (specific) indiscriminate poisoning; (ii) $\alpha = -1$ and $\beta = -1(1)$ for error-specific (generic) targeted poisoning; (iii) $\alpha = -1$ and $\beta = -1(1)$ for error-specific (generic) backdoor poisoning.

In conclusion, although backdoor, indiscriminate and targeted attacks are designed to cause distinct security violations, they can be formulated under a unique bilevel optimization program. Therefore, as we will explore in Sec. 3.3.5, solutions for optimizing bilevel optimization programs fast can pave the way towards developing novel effective and stealthy poisoning attacks capable of mitigating the scalability limit of current strategies.

3.3.5 Development Timeline

In this section we discuss the intertwined historical development of attacks, represented in Fig. 3.5, highlighting the respective milestones and development over time. To the best of our knowledge, the first example of indiscriminate poisoning was developed in 2006 by Perdisci et al. [218], Barreno et al. [16], and Newsome et al. [202] in the computer security area. Such attacks, as well as subsequent attacks in the same area [142; 227], were based on heuristic approaches to mislead application specific ML models, and there was not a unifying mathematical formulation describing them. It was only later, in 2012, that indiscriminate poisoning against machine learning was formulated for the first time as a bilevel optimization [292], to compute optimal label-flip poisoning attacks. Since then, indiscriminate poisoning has been studied under two distinct settings, i.e., assuming either (i) that a small fraction of training samples can be largely perturbed [27; 197; 73]; or (ii) that all training points can be slightly perturbed [191; 93; 94].

Targeted and backdoor poisoning attacks only appeared in 2017, and interestingly, they both started from different strategies. Targeted poisoning started with the bilevel formulation in Koh and Liang [144], but evolved in more heuristic approaches, such as feature collision [238; 323; 109]. Only recently, targeted poisoning attacks were reformulated as bilevel problems, given the limitation of the aforementioned heuristic approaches [129; 101]. Backdoor poisoning started with the adoption of patch [108; 173] and functional [175; 204] triggers. However, in the last years, such heuristic choices have been put aside, and backdoor attacks are getting closer and closer to the idea of formulating them in terms of a bilevel optimization, not only to enhance their effectiveness, but also their ability to bypass detection [229; 251].

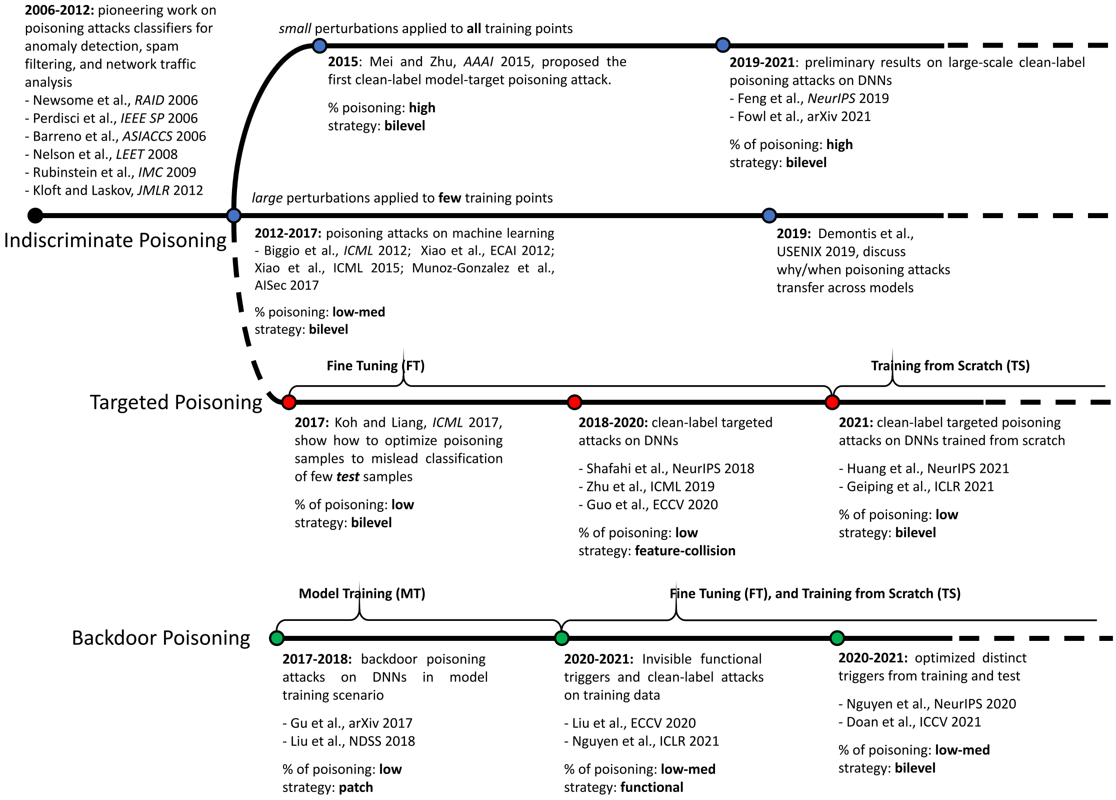

2006-2012: pioneering work on poisoning attacks classifiers for anomaly detection, spam filtering, and network traffic analysis
- Newsome et al., RAID 2006
- Perdisci et al., IEEE SP 2006
- Barreno et al., ASIACCS 2006
- Nelson et al., LEET 2008
- Rubinstein et al., IMC 2009
- Kloft and Laskov, JMLR 2012
small perturbations applied to all training points
2015: Mei and Zhu, AAAI 2015, proposed the first clean-label model-target poisoning attack.
% poisoning: high
strategy: bilevel
2019-2021: preliminary results on large-scale clean-label poisoning attacks on DNNs
- Feng et al., NeurIPS 2019
- Fowl et al., arXiv 2021
% of poisoning: high
strategy: bilevel
large perturbations applied to few training points
Indiscriminate Poisoning
2012-2017: poisoning attacks on machine learning
- Biggio et al., ICML 2012; Xiao et al., ECAI 2012; Xiao et al., ICML 2015; Munoz-Gonzalez et al., AISec 2017
% poisoning: low-med
strategy: bilevel
2019: Demontis et al., USENIX 2019, discuss why/when poisoning attacks transfer across models
Fine Tuning (FT)
Training from Scratch (TS)
Targeted Poisoning
2017: Koh and Liang, ICML 2017, show how to optimize poisoning samples to mislead classification of few test samples
% of poisoning: low
strategy: bilevel
2018-2020: clean-label targeted attacks on DNNs
- Shafahi et al., NeurIPS 2018
- Zhu et al., ICML 2019
- Guo et al., ECCV 2020
% of poisoning: low
strategy: feature-collision
2021: clean-label targeted poisoning attacks on DNNs trained from scratch
- Huang et al., NeurIPS 2021
- Geiping et al., ICLR 2021
% of poisoning: low
strategy: bilevel
Model Training (MT)
Fine Tuning (FT), and Training from Scratch (TS)
Backdoor Poisoning
2017-2018: backdoor poisoning attacks on DNNs in model training scenario
- Gu et al., arXiv 2017
- Liu et al., NDSS 2018
% of poisoning: low
strategy: patch
2020-2021: Invisible functional triggers and clean-label attacks on training data
- Liu et al., ECCV 2020
- Nguyen et al., ICLR 2021
% of poisoning: low-med
strategy: functional
2020-2021: optimized distinct triggers from training and test
- Nguyen et al., NeurIPS 2020
- Doan et al., ICCV 2021
% of poisoning: low-med
strategy: bilevel

Fig. 3.5: *Timeline for indiscriminate (blue), targeted (red) and backdoor (green) data poisoning attacks on machine learning. Related work is highlighted with markers of the same color and connected with dashed lines to highlight independent (but related) findings.*

3.3.6 Transferability of Poisoning Attacks

Transferability is a characteristic of attacks to be effective even against classifiers of which the attacker does not have complete knowledge. The term transferability was first investigated for adversarial examples in [105; 211; 210]. In case of limited knowledge (i.e., black-box attacks), the attacker can use surrogate learners or training data to craft the attack and transfer it to mislead the unknown target model. Nevertheless, the first to introduce the idea of surrogates for data poisoning attacks were Nelson et al. [200] and Biggio et al. [27]. The authors claimed that if the attacker does not have exact knowledge about the training data, they could sample a surrogate dataset from the same distribution and transfer the attack to the target learner. In subsequent work, Muñoz-González et al. [197] and Demontis et al. [73] also analyzed the transferability of poisoning attacks using surrogate learners, showing that matching the complexity of the surrogate and the target model enhances the attack effectiveness. Transferability has also been investigated when considering surrogate objective functions. More concretely, optimizing attacks against a smoother objective function may find effective, or even better, local optima than the ones of the target function [182; 73; 144; 210]. For example, optimizing a non-differentiable loss can be harder; thus, using a smoothed version may turn out to be more effective [144]. More recently, Suciu et al. [256] showed that the attacker could leverage transferability even when the attacker has limited knowledge about the feature representation, reducing the attack effectiveness. However, Zhu et al. [323] and Aghakhani et al. [2] independently hypothesize that the stability of *feature collision* attacks is compromised when the feature representation in the representation space is changed. To mitigate this problem, they craft poisoning samples to attack an ensemble of models, encouraging their transferability against multiple networks.

3.4 Defenses Against Poisoning

Many defenses have been proposed to mitigate poisoning attacks. This section discusses each of the six defense classes we identified in our survey paper [63]. For each group, we expose the general underlining strategy adopted to mitigate or remove the influence of poisoning attacks.

Training Data Sanitization. These defenses aim to identify and remove poisoning samples *before training*, to alleviate the effect of the attack. The underlying rationale is that poisoning samples must be *different* from the rest of the training points to be effective; otherwise, they would have no impact on the training process. Accordingly, poisoning samples typically exhibit an outlying behavior with respect to the training data distribution, which enables their detection. Detection can also be eased by taking into account features and labels to find anomalies internal to the data classes.

Robust Training. Another possibility to mitigate poisoning attacks is *during training.* The underlying idea is to protect the system *during training* by designing a learning algorithm that limits the influence of malicious samples and alleviates the influence of the poisoning attack. In our paper [59], described in Chapter 5, we show that a wise choice of the model's hyperparameters related to regularization can significantly limit the influence of poisoning samples performance.

Model Inspection. Starting with model inspection, we discuss groups of defenses operating before the model is deployed. The approaches in these groups mitigate only backdoor and targeted attacks. In model inspection, we determine for a given model whether a backdoor is implanted or not. Such defenses aim at testing whether the model behaves somehow *unusual*: it will rely on normally irrelevant features. Thus, outlier detection mechanisms are used on top of interpretability techniques or latent data representations.

Model Sanitization. Once a backdoored model is detected, the question becomes how to sanitize it. Model sanitization often involves pruning, (re-)training, or fine-tuning to sanitize the model to remove the influence of poisoning and restore the model's prediction accuracy. However, sanitizing the model might be impossible if the model is provided as a service accessible only via queries.

Trigger Reconstruction. This category of defenses aims to reconstruct the implanted trigger as an alternative to model sanitization. Many techniques leverage the fact that a trigger changes the classifier's output reliably. The classifier ignores other features and only relies on the backdoor trigger. Such a stable output also enables to reformulate trigger reconstruction as an optimization problem whose solution is a pattern that leads to reliable misclassification of a batch of input points.

Test Data Sanitization. As the name suggests, this is the only group of defenses operating during *test time*, where the defender attempts to sanitize malicious test inputs. These techniques try to identify crucial parts of the input and then mask these to identify whether they are adversarial or not.

3.4.1 Matching Poisoning Attacks and Defenses

We further match attack strategies and defenses at training and test time in Table 3.2. We match poisoning attack strategies and defenses by reporting in each cell the defensive papers that evaluate their effectiveness against the corresponding attack strategy. We mark with ✗ the cells corresponding to trigger reconstruction for targeted and indiscriminate attacks and test data sanitization for indiscriminate poisoning, since no trigger is exploited and test data are clean in such attacks.

Despite the large body of work on defenses, there are still unresolved challenges. It immediately becomes evident that not all attacks sparked equally many mitigations. While about fifty defenses exist for backdoor attacks using patch triggers, only eleven defenses have been considered against semantic triggers and a few against bilevel attacks. Indeed, there are still no defenses against indiscriminate clean-label bilevel attacks and just a few defenses against recent backdoor bilevel attacks. This deficiency is not coincidental, as bilevel attacks have been developed more recently, plus they seem more effective and stealthy, making their detection even more difficult than in past work. Furthermore, few defenses (only about one-sixth) have been evaluated against different types of triggers, and although their application may be intriguing, there are no or a few model inspection or sanitization approaches in the direction of indiscriminate, targeted poisoning.

Table 3.2: *Matching poisoning attack strategies and defenses. For each defense, we depict on which attack strategy (as defined in Sec. 3.2) the defense was evaluated. We mark cells with ▬ if the corresponding defense category have not been investigated so far for the corresponding attack. Conversely, we mark cells with ✗ if corresponding defense has no sense and cannot be applied.*

Attack				Defenses					
				Training Time					Test Time
	δ	t	Clean Label	Training Data Sanitization	Robust Training	Model Inspection	Model Sanitization	Trigger Reconstruction	Test Data Sanitization
Indiscriminate	LF	-		[214; 153; 263]	[122; 158; 26; 72; 225; 278; 48]	▬	▬	✗	✗
	BL	-		[96; 253]	[201; 25; 135; 182]	▬	▬	✗	✗
	BL	-	✔	▬	▬	▬	▬	✗	✗
Targeted	BL	-		[96]	▬	▬	▬	✗	▬
	FC	-	✔	[219; 239; 302]	[100; 34; 166; 122]	[325; 264]	[325]	✗	▬
	BL	-	✔	[239; 302]	[100; 34; 33]	▬	▬	✗	▬
Backdoor	T^P	T^P		[269; 289; 286; 273; 118; 239]	[304; 280; 33; 166; 135; 100; 128; 258; 80]	[264; 44; 145; 317; 242; 46; 300; 12; 288; 250; 110; 290; 127]	[169; 307; 304; 319; 222; 46; 164; 324; 284; 308]	[111; 276; 174; 271; 222; 53; 79; 289; 110; 290; 127]	[75; 163; 307; 272; 276; 57; 97; 233; 220; 273]
	T^S	T^S		▬	[128; 239]	[242; 110]	[172; 169; 307; 308]	[110]	[172; 272]
	T^F	T^F		[289; 118]	[280; 166; 128]	[264; 242; 130; 300; 250; 288; 127]	[324; 307; 164; 284; 308]	[287; 324; 174; 288; 289; 127]	[307; 97]
	FC	T^P	✔	[118]	[100; 34; 166; 128; 302]	[325; 110]	[325; 324; 164; 284]	[110]	[163]
	BL	T^F		▬	▬	[242; 110]	[284]	[324]	[307]
	BL	T^P	✔	▬	[302]	▬	▬	▬	▬

3.5 Poisoning in Other Domains

While in this survey we focus on poisoning ML in the context of supervised learning tasks, and mostly in computer-vision applications, it is also worth remarking that several poisoning attacks and defense mechanisms have been developed also in the area of federated learning [11; 22; 36; 119; 259; 266; 296; 311; 312; 313; 321], regression learning [74; 92; 133; 168; 196; 281], reinforcement learning [7; 14; 19; 90; 114; 131; 141; 181; 223; 277; 298; 315], and unsupervised clustering [29; 31; 64; 142; 227] or anomaly detection [67; 227] algorithms. Furthermore, notable examples of poisoning attacks and defenses have also been shown in computer-security applications dealing with ML, including spam filtering [25; 74; 96; 200; 214], network traffic analysis [227], and malware detection [218; 237; 263], audio [1; 143; 169; 173; 300] and video analysis [269; 320], natural language processing [45; 51; 173; 220; 316], and even in graph-based ML applications [32; 170; 318; 326]. While, for the sake of space, we do not give a more detailed description of such research findings in this survey, we do believe that the systematization offered in our work provides a useful starting point for the interested reader to gain a better understanding of the main contributions reported in these other research areas.

3.6 Concluding Remarks

The increasing adoption of data-driven models in production systems demands a rigorous analysis of their reliability in the presence of malicious users aiming to compromise them. Within this chapter, we systematize a broad spectrum of data poisoning attacks according to our modeling framework, and we exploit such categorization to match defenses with the corresponding attacks they prevent. Moreover, we provide a unified formalization for poisoning attacks via bilevel programming, and we traced the historical development of data poisoning literature since the early developments dating back to more than 20 years ago, identifying the ongoing research directions. We finally reviewed the literature on the transferability of data poisoning, which is deemed to be a promising way to stage poisoning attacks when the attacker has limited knowledge of the victim system.

In conclusion, responding to our research question, "How can we categorize/distinguish poisoning attacks against ML", we would say **"poisoning attacks can be categorized according to our attack framework (see Sec. 3.2) considering their threat model, characterized by the attacker's goal, knowledge, capability, and strategy."**. The attacker's goal and knowledge define the kind of violation the attacker aims to cause with a poisoning attack and their knowledge about the target system (e.g., knowledge of the ML model or training data). The capability defines the attacker's influence in the model training pipeline, i.e., attacking the training data (fine-tuning and training-from-scratch) or controlling the training process (model-training). Each scenario brings a different challenge the attacker has to address to stage effective and stealthy poisoning attacks. Finally, the strategy defines how the attacker tampers with the training data to stage the attack. Simple strategies may be faster to be staged, requiring a lower amount of computational resources, while more sophisticated ones can be more effective and stealthy in case of defender inspection but may bring higher computational costs. These characteristics are essential for a good categorization of state-of-the-art poisoning attacks. Comparison performance of attacks under different assumptions may therefore bring unfair conclusions. For example, a limitation in knowledge or capability may represent a more challenging scenario for the attacker, where it is reasonable to have a loss of performance compared to other works assuming higher control on the victim system.

We believe our contribution can help clarify what threats an ML system may encounter in adversarial settings and encourage further research developments in deploying trustworthy systems even in the presence of data poisoning threats.

Chapter 4

Improving Scalability of Data Poisoning

RQ#2

Can we make poisoning attacks scalable?

In this Chapter we respond to our RQ#2, which examines the computational complexity involved in staging data poisoning attacks and tries to reduce it by making them scalable in practice. Questioning this point is critically relevant for two reasons. First, it allows us to understand whether state-of-the-art attacks pose a real threat to real-world systems. For an attack to be meaningful to the attacker, it must have a reasonable cost, which translates into the amount of time and hardware required to carry it out. If the attack does not scale, it may not be considered a relevant threat against large systems. The second reason is that it is fundamental to understand how far we are in realizing scalable benchmarks to test the robustness of ML models against poisoning. A benchmark indeed requires executing a series of attacks and measuring the model's performance against them. The underlying assumption is, however, that these attacks can be feasible executable under various conditions. For example, the benchmark must run in a feasible amount of time, even when increasing the dataset or the model's size. Analyzing and minimizing the computational costs of poisoning samples is fundamental both in the attacker and defender directions. Therefore, this Chapter explores the scalability issues of poisoning attacks and sheds light on how to bridge them under specific assumptions using faster heuristic approaches.

Underlying Problem. As seen in Sec. 3.3, mathematically speaking, poisoning attacks require solving a bilevel optimization problem where the outer problem consists of minimizing or maximizing the accuracy on a validation set while reducing, in the inner problem, the accuracy of the poisoned training set [101; 197]. However, solving this problem for each poisoning sample can be computationally demanding for the attacker, especially when dealing with large models (e.g., DNNs) and datasets. In [197] the authors noted that the solution of the bilevel program requires the inversion of the Hessian matrix whose algorithmic complexity is cubic with respect to the number of model parameters. This aspect makes the usage of this algorithm computationally prohibitive for various practical settings. Nonetheless, according to the nature of the violation and the ambitions that the attacker desires, the attack cost can drastically be reduced.

Related Work. Thanks to some heuristics, e.g., feature collision [238], gradient-alignment [101], the attacker can also stage targeted poisoning attacks efficiently against systems trained on large-scale datasets. Effective backdoor poisoning attacks can be staged by simply implanting the backdoor trigger into a few training samples to reach the attacker's desideratum [15; 108; 175]. Conversely, *indiscriminate poisoning* attacks have been limited in scope and success due to the com-

putational requirements [27; 73; 134; 197]. Muñoz-González et al. [197] approximated the problem by adopting a back-gradient optimization technique taken from the hyperparameter optimization domain. Similarly, Jagielski et al. [134] linearly approximated the bilevel formulation to reduce computational costs further. Yang et al. [301] adopted a generative model for crafting poisoning samples. Once the model is trained, the attacker exploits it to fast generate new poisoning samples. The advantage of their approach is that model is trained to intrinsically approximate the bilevel formulation, locating poisoning samples in the model's vulnerable region. Finally, Demontis et al. [73] observed that the poisoning samples, crafted to poison a target model, are effective even against other systems. Thus an attacker could generate attacks with a smaller model, i.e., less computationally demanding, and transfer them to the larger target one. However, they all remain more computationally demanding than heuristic integrity attacks because they do not follow the complex bilevel attack strategy [238; 101].
To the best of our knowledge, no existing heuristic approaches have been proposed to scale indiscriminate poisoning, and previous heuristics for targeted and backdoors are unsuitable. Feature collision allows to stage of an attack that misclassifies only a few target samples, so it does not generalize as required for indiscriminate, which aims at causing a Denial of Service (DoS). Gradient-alignment has been recently adopted in [94], but its threat model is fairly limited as it assumes the attacker to perturb the entire training set, and it is an open question whether the attack generalizes to any test data.

Contributions and Outline. While we have heuristic approaches for scaling for targeted and backdoors, the same does not apply to indiscriminate attacks. This implies an initial and partial answer to our research question, "yes, we can scale at least targeted and backdoor poisoning attacks thanks to promising heuristic approaches". Regarding indiscriminate poisoning attacks, in our work [60], explained in the remaining of this Chapter, we observed that in particular conditions, namely, where the target model is linear (the "nut"), the usage of computationally costly procedures (the "hammer") for crafting poisoning samples could be avoided. We thus propose in Sec. 4.1 our counter-intuitive but efficient heuristic that contaminates the training set so that the target system's performance is highly compromised. We further suggest a re-parameterization trick in Sec. 4.1 that aims to decrease our attack's computational demand by decreasing the number of variables to be optimized. Finally, in Sec. 4.2 we demonstrate that, under the considered settings, our framework achieves comparable, or even better, performances in terms of the attacker's objective while being significantly more computationally efficient. We conclude this Chapter in Sec. 4.3 highlighting how our work has helped to make discriminate attacks more scalable under certain assumptions while emphasizing that there is still a long way to go to make attacks practical in more challenging environments.

4.1 BetaPoisoning

Notation. We here recap the notation used in the remaining of this Chapter. Further details are reported in Chapter 3, and a list of symbols is given on page v.

Feature and label spaces are denoted in the following with $\mathcal{X} \subseteq \mathbb{R}^{\mathsf{d}}$ and $\mathcal{Y} \in \mathbb{R}$, respectively, with d being the dimensionality of the feature space. The attacker collect a training and a surrogate dataset that will be used to craft the attack. We denote them as $\mathcal{D}'$ and $\mathcal{V}$. We define $\mathcal{V}^{y_t} = \{\boldsymbol{x} | (\boldsymbol{x}, y_t) \in \mathcal{V}\}$ a subset of the surrogate dataset with samples of class y_t. Note that these sets include samples along with their labels. We define with $L(\mathcal{V}, \boldsymbol{\theta})$ the loss incurred by the classifier $f : \mathcal{X} \to \mathcal{Y}$, parametrized by $\boldsymbol{\theta}$, on the surrogate dataset $\mathcal{V}$. $\mathcal{L}(\mathcal{D}', \boldsymbol{\theta})$ is used to represent the regularized loss optimized by the classifier during training.

Given the scalability gap we identified between integrity and indiscriminate poisoning attack, we propose a novel heuristic attack that aims at bridging the gap when certain conditions are satisfied.

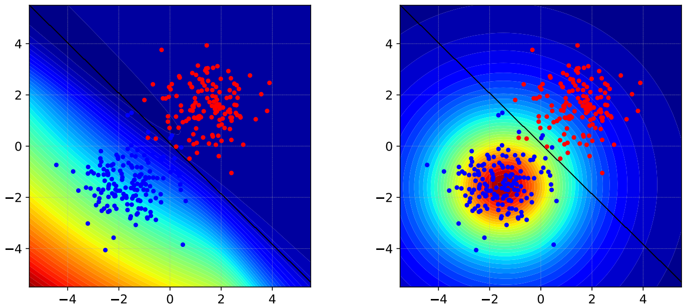


Fig. 4.1: *(Left) Attacker's objective function with bilevel formulation [27]. (Right) BetaPoisoning objective function, described in Eq.* (4.1)*. Red regions represent zones where the objective function is high. The black line is the decision boundary of a logistic regression classifier.*

The resulting poisoning attack, called BetaPoisoning, exploits linear classifiers' limits when dealing with noisy-labeled samples [35]. In particular, the attacker poisons the target distributions y_t with sample $\boldsymbol{x}_p$ by maximizing the likelihood $P(\boldsymbol{x}_p|y_t)$, making the dataset no longer linearly separable. Interestingly, it does not need access to the training set, and it does not need to have access to the target model while crafting the poisoning samples.

We illustrate the idea behind the proposed approach with an example in Fig. 4.1, which visualizes the difference between our objective function and the bilevel one in Eqs. (3.1)-(3.2). To create an easily understandable example, we consider a linearly separable two-dimensional dataset in which each class follows a Gaussian distribution. Based on the bilevel formulation for indiscriminate poisoning illustrated in Sec. 3.3, the red area shows that poisoning samples should be located in the bottom-left region to obtain the highest validation error (left plot in Fig. 4.1). However, bilevel-based poisoning algorithms are computationally demanding. Conversely, the objective function of our heuristic approach, shown in the right plot of Fig. 4.1, suggests locating the poisoning samples in the space region with the highest density of training samples.

This is a counter-intuitive solution because the optimal region is quite different from the one obtained optimizing the bilevel problem. The optimum suggested by our heuristic approach is distant from the optimal one. Nonetheless, optimizing the proposed objective function allows us to achieve good results under the considered settings. Similarly to label-flip attacks [26; 214; 292; 294] the attacker poisons the target distributions making the learning task harder by flipping the labels of training samples. Notwithstanding, these points may be located in non-dense regions; hence even if we flip their labels, they may not significantly reduce the classifier's performance. We thus formulate BetaPoisoning, whose objective is reported in Eqs (4.1)-(4.2) in order to look for the regions with the highest density of samples and put poisoning samples inside of them.

$$\arg\max_{\boldsymbol{x}_p} \quad P(\boldsymbol{x}_p|y_t) \tag{4.1}$$

$$\text{s.t.} \quad \boldsymbol{x}_{\text{lb}} \preceq \boldsymbol{x}_p \preceq \boldsymbol{x}_{\text{ub}} \tag{4.2}$$

where Eq. (4.2) defines box-constraints on the poisoning sample's feature values. The likelihood $P(\boldsymbol{x}_p|y_t)$ is estimated using a Gaussian Kernel Density Estimator (KDE), where the bandwidth parameter h is chosen equal to the average distance between all possible pairs of samples in $\mathcal{V}^{y_t}$ [29]. This heuristic attack considers only the data distribution; therefore, conversely to [27; 73], we do not need to know the model's parameters.

$$P(\boldsymbol{x}_p|y_t) = \frac{1}{\|\mathcal{V}^{y_t}\|} \sum_{\boldsymbol{x}\in\mathcal{V}^{y_t}} \exp\left(-\frac{\|\boldsymbol{x}_p - \boldsymbol{x}\|^2}{h}\right) \tag{4.3}$$

4.1.1 Poisoning Re-parametrization

Crafting a single poisoning sample requires optimizing d-dimensional variables. When dealing with a huge dataset d can be very large, thus increasing the complexity of optimizing them and the computational costs. Although BetaPoisoning does not solve a bilevel optimization program, we further decrease its computational complexity by adopting a re-parametrization trick that aims to reduce the number of variables to optimize. We impose that our poisoning samples are obtained as a linear combination of other samples in the dataset. We define $\mathcal{S} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_k\}$ a random subset of samples with label y_t in $\mathcal{V}$. We define with $k = |S| \ll \mathsf{d}$ the number of samples, named prototypes, in $\mathcal{S}$. Given a random set of samples $\mathcal{S}$ and coefficients $\boldsymbol{\beta} \in \mathbb{R}^k$, the corresponding poisoning sample $\boldsymbol{x}_p$ is obtained with:

$$\boldsymbol{x}_p = \psi(\boldsymbol{\beta}, S) = \sum_{\boldsymbol{x}_i \in \mathcal{S}} \beta_i \boldsymbol{x}_i \tag{4.4}$$

We can thus reformulate our optimization problem in terms of $\boldsymbol{\beta}$ coefficients in the following way:

$$\arg\max_{\boldsymbol{\beta}} \quad P(\psi(\boldsymbol{\beta}, S)|y_t) \tag{4.5}$$

$$\text{s.t.} \quad \boldsymbol{x}_{\text{lb}} \preceq \psi(\boldsymbol{\beta}, S) \preceq \boldsymbol{x}_{\text{ub}} \tag{4.6}$$

Once the optimal $\boldsymbol{\beta}$ coefficients are optimized, we can easily reconstruct the resulting poisoning sample with $\psi(\boldsymbol{\beta}, S)$.

Algorithm 4: BetaPoisoning

```
   Input: $\mathcal{V} \in \mathbb{R}^{n \times \mathsf{d}}, y_t, k, \boldsymbol{x}_{lb}, \boldsymbol{x}_{ub}$
   Output: Poison sample $\boldsymbol{x}_p \in \mathbb{R}^d$
 1
 2 $\mathcal{S}$ = sample_random_prototypes($\mathcal{V}, y_t$)
 3 $\boldsymbol{\beta}$ = init_beta(k)
 4 repeat
 5     $\boldsymbol{x}_p$ = clip($\psi(\boldsymbol{\beta}, S)$, $\boldsymbol{x}_{lb}$, $\boldsymbol{x}_{ub}$)
 6     $p = P(\boldsymbol{x}_p|y_t)$
 7     $\boldsymbol{\beta} = \boldsymbol{\beta} + \alpha \nabla_{\boldsymbol{\beta}} p$
 8     $\boldsymbol{x}_p$= clip($\psi(\boldsymbol{\beta}, S)$, $\boldsymbol{x}_{lb}$, $\boldsymbol{x}_{ub}$)
 9 until Poisoning point remains unchanged (convergence)
10 return $\boldsymbol{x}_p$
```

Algorithm. A pseudo-code description of our attack can be found in Algorithm 4. We get in input a surrogate dataset $\mathcal{V}$, the target class y_t, the number of prototypes k, and the box constraints $(\boldsymbol{x}_{lb}, \boldsymbol{x}_{ub})$. We note that the attacker may choose y_t and y_p according to some strategy or prior knowledge about the application context. In our setting, we assume that the attacker chose y_t and y_p randomly during the rest of this work. In line 2 we randomly sample a subset of prototypes from $\mathcal{V}$ and we initialize the corresponding $\boldsymbol{\beta}$ in line 3. Initialization for the $\boldsymbol{\beta}$ coefficients is done by randomly sampling values in $[0, 1]^{\mathsf{d}}$. In line 5 we construct the poisoning point and clip it to preserve box constraints. In line 6 we estimate the likelihood $P(\boldsymbol{x}_p|y_t)$ with the KDE in Eq. (4.3). We then update the $\boldsymbol{\beta}$ coefficients through a gradient ascend step, with a learning rate $\alpha = 0.01$ and we clip it to preserve box-constraints in Lines 7-8. The process described from Line 5 to 8 is repeated until a certain stop condition is reached, i.e., if the attacker's objective function $P(\boldsymbol{x}_p|y_t)$, in two consecutive iterations, does not change more than a pre-defined threshold. In our experiments, we set this threshold to $1e - 05$.

4.2 Experimental Analysis

In this section, we evaluate the effectiveness of our attack for poisoning linear classifiers. We first consider poisoning against binary models, as done in [27; 73], and then extend our analysis to the multi-class setting. We then propose a time execution comparison between our approach and state-of-the-art bilevel poisoning attacks. Finally, we evaluate the performance of our attack when varying the number of prototypes k, suggesting that even a few of them are sufficient for staging effective attacks. We run our experimental evaluation five times, and we report the mean accuracy and the corresponding standard deviation. Code for all experiments can be found at `github.com/Cinofix/beta_poisoning`.

4.2.1 Experimental Setup

Datasets. In our experimental analysis we use two publicly available datasets, namely MNIST [157] and CIFAR10 [147]. MNIST is a dataset for digits recognition containing $70,000$ gray-scale 28×28px images divided in 10 digit classes (from 0 to 9). The CIFAR10 dataset [147] contains $60,000$ color images of 32×32 pixels equally distributed in 10 classes. Being datasets of images, each pixel takes values in the range $[0, 255]$. We normalize pixels by dividing their values by 255, using them as our features.

For binary classification with MNIST we consider pairs 9 vs. 8 and 4 vs. 0 (as in [27]). We randomly sample $400, 1000$, and 1000 samples for training, validation, and test set for each pair of digits. This setting is similar to the one reported in [27], but we increased the number of training and validation samples. Similarly, for CIFAR10 we consider the two pairs of classes with the highest accuracy on untainted dataset, *frog* vs. *ship* and *horse* vs. *ship*. We randomly sample $300, 500$, and 1000 images for each of them to build our training, validation, and test set, respectively.
When poisoning non-binary classifiers we consider two random triplet of classes from MNIST ($\{3, 7, 5\}$ and $\{9, 4, 0\}$). We compose training, validation, and test sets for each triplet by randomly sampling 400, 1000, and 1000 images. We test the effectiveness of poisoning against an SVM with regularization terms $C = 1$ and $C = 100$.

Models and Training Phase. We train and test the robustness of a linear support vector machine (SVM) and a logistic regression classifier (LC) under different regularization levels for both datasets. Further details about these models are given in Sec. 2.1.1. We also test the effectiveness of our attack when changing the hyperparameters of the victim models, namely C, which regulate their generalization capacity.

Poisoning Attacks. We compare our attack against the random label flips attack [26], and the bilevel attacks proposed in [27] (for SVM) and [73] (for LC). Since the two bilevel attacks are suited for binary classification problems, we compare our attack only against the label flips attack when attacking multi-class classifiers. The effectiveness of all the compared algorithms is assessed on a test set, never seen during the optimization. Notably, both BetaPoisoning and label flip do not need to know the training set as required for [27] and [73], but they exploit the surrogate dataset to stage the attack.

4.2.2 Experimental Results

Poisoning Binary Classifiers. We reported in Fig. 4.2 and Fig. 4.3 the results obtained for SVM and LC with regularization parameter $C = 1$ and $C = 100$. We can notice a steady growth of the attack effectiveness with the increasing fraction of poisoning points added to the training set. In particular, we observe that when the penalty term C increases, the target models become less robust against poisoning attacks. These results have also been observed in [73], where the authors state that strongly regularized classifiers tend to have smaller input gradients, i.e., they learn smoother functions that are more robust to attacks. Notably, the performance obtained by our framework in terms of the attacker's objective is comparable with [27], or even better when the regularization of the target models decreases. Table 4.1 reports the computational costs needed to run the three poisoning algorithms when the percentage of attack points in the training set is 20%.

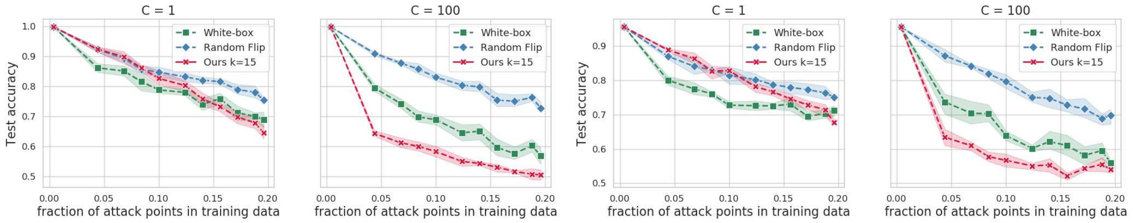


Fig. 4.2: *Accuracy on the test set for SVM, with regularization $C = 1$ and $C = 100$, under poisoning attack. (Left) results for MNIST pair 4 vs. 0, (right) results for pair 9 vs. 8. In green the performance for [27].*

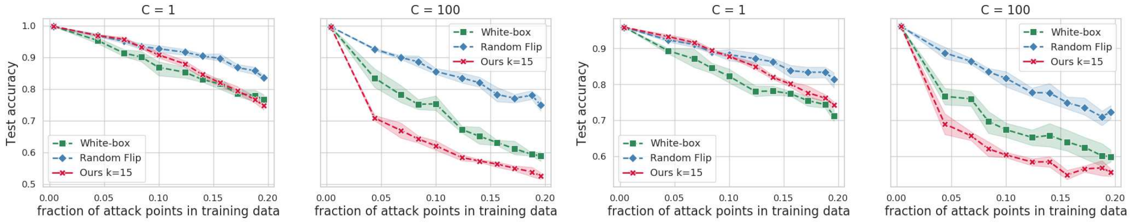


Fig. 4.3: *Accuracy on the test set for LC, with regularization $C = 1$ and $C = 100$, under poisoning attack. (Left) results for MNIST pair 4 vs. 0, (right) results for pair 9 vs. 8. In green the performance for [73].*

We report in Fig. 4.4 and Fig. 4.5 the results of poisoning against linear SVM and LC with different regularization strengths for CIFAR10. Notably, the obtained results are consistent with those described for MNIST. This means that our poisoning samples are effective even on large datasets. Notably, the computational gap for LC, reported in Table 4.2, is significantly increased, favoring our method.

Poisoning Multi-class Classifiers. In Fig. 4.6 we report the results of BetaPoisoning and label flips attack [26] against multi-class classifiers on MNIST triplets. The comparison with the bilevel attack in [27] is missing as its implementation is only suited for binary classification problems, while the comparison results with the label flip attack prove that the proposed *counter-intuitive* attack is effective even against multi-labels classification tasks. Moreover, even in this case, when the regularization term C increases, the robustness to poisoning decreases significantly. Notably, the two random triplets' performance has the same trend, confirming the goodness and stability of our approach to random selection.

Time Comparison. We now analyzes the computational costs provided by our approach and the bilevel poisoning algorithms [27; 73]. We run our experiments for MNIST and CIFAR10 on a Intel® Xeon® Processor E5-2690 v3.

Table 4.1 shows the results obtained for the two MNIST pairs when generating 100 poisoning samples(20% of the training set is poisoned). As we expected, the comparison of the attack times shows the proposed algorithm's reduced computational cost, highlighting a significant gap. Even if, as shown in Figs. 4.2-4.6, the attack effectiveness is comparable, we can craft poisoning points more efficiently without solving a complex bilevel optimization problem. Moreover, our experimental analysis highlights that the running time of the attack that solves the bilevel optimization problem is strongly influenced by different factors, such as the learning algorithm and the regularization strength. That attack is slower against LC than against SVM and is less computationally expensive against strongly regularized classifiers. In comparison, the running time of the proposed BetaPoisoning is almost constant.

Similarly, Table 4.2 reports the computational costs for the two CIFAR10 pairs when generating

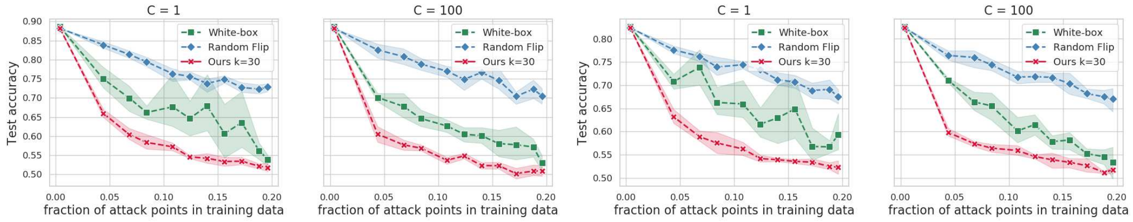


Fig. 4.4: *Accuracy on the test set for SVM, with regularization $C = 1$ and $C = 100$, under poisoning attack. Results for CIFAR10 pair frog vs. ship (left) and pair horse vs. ship (right). In green, the performance for [27].*

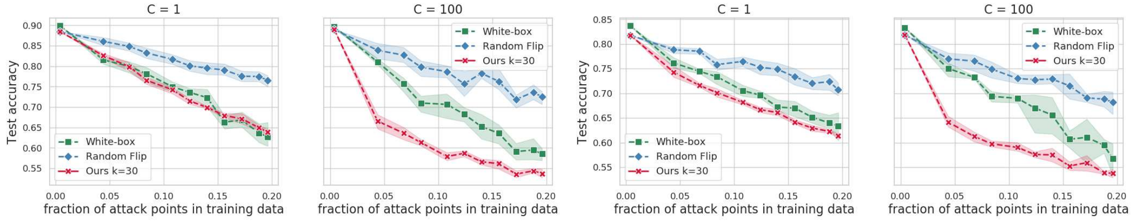


Fig. 4.5: *Accuracy on the test set for LC, with regularization $C = 1$ and $C = 100$, under poisoning attack. Results for CIFAR10 pair frog vs. ship (left) and pair horse vs. ship (right). In green, the performance for [73].*

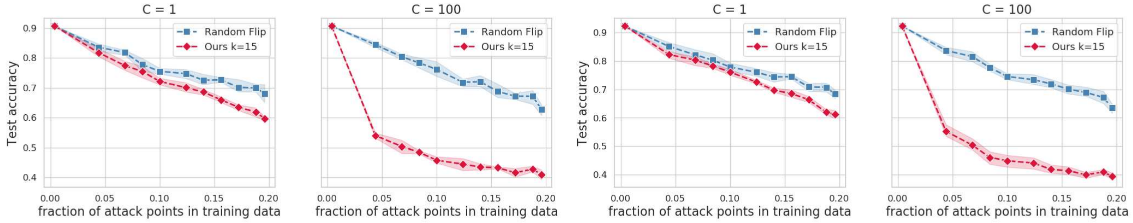


Fig. 4.6: *Accuracy on the test set for SVM, with regularization $C = 1$ and $C = 100$, under poisoning attack. The two on the left are obtained on triplet $\{3, 7, 5\}$, and the two on the right are obtained for triplet $\{9, 4, 0\}$.*

75 poisoning samples (20% of the training set is poisoned). Even in this case, our algorithm best performs with significant evidence against SVM and LC. We notice that the computational performance gap between SVM and LC is higher than the ones exhibited in the MNIST dataset. On the CIFAR10 dataset, when the classifier's complexity increases, the performance for [27] and [73] improves, but not enough to bridge the gap with our algorithm.

Ablation Study. We further assess the effectiveness of BetaPoisoning by varying the number of prototypes k. We remark that the number of prototypes corresponds exactly with the number of coefficients $\boldsymbol{\beta}$ to be optimized during the learning process. We use the same configuration detailed before for poisoning binary classifiers trained with MNIST (4 vs. 0) and CIFAR10 (frog vs. ship). We let the number of prototypes vary from 2 to 30 and analyze our poisoning attack's performance against a linear SVM with the regularization term $C = 1$.
Fig. 4.7 shows that the number of prototypes chosen may significantly influence our algorithm's performance. In particular, when increasing k, our framework seems to create more powerful poisoning points. We observe that for a smaller dataset like MNIST, 15 prototypes are sufficient. Conversely, the optimal number of prototypes for a more complex dataset like CIFAR10 is 30. This reduction represents a significant improvement; indeed if we consider Eqs. (3.1)-(3.2) and Eq. (4.1), the number of variables to optimize is equal to the sample's dimension. Thanks to our

Table 4.1: *Computational cost comparison between poisoning algorithms against SVM and LC for MNIST 4-0 and 9-8.*

Model	Dataset	Generator	Time in s	
			C=1	C=100
SVM	4-0	[27]	123.44 ± 11.59	148.99 ± 44.66
	4-0	**BetaPoisoning**	**11.06 ± 0.60**	**11.40 ± 1.78**
	9-8	[27]	132.39 ± 24.64	168.17 ± 53.26
	9-8	**BetaPoisoning**	**11.09 ± 0.25**	**11.27 ± 0.22**
LC	4-0	[73]	261.46 ± 30.69	459.57 ± 22.60
	4-0	**BetaPoisoning**	**10.70 ± 0.55**	11.06 ± 1.64
	9-8	[73]	285.09 ± 31.39	458.22 ± 15.55
	9-8	**BetaPoisoning**	**11.17 ± 0.21**	**11.45 ± 0.48**

Table 4.2: *Computational cost comparison between poisoning algorithms against SVM and LC for CIFAR10 frog-ship and horse-ship.*

Model	Dataset	Generator	Time in s	
			C=1	C=100
SVM	frog-ship	[27]	115.72 ± 24.72	40.21 ± 2.35
	frog-ship	**BetaPoisoning**	**06.95 ± 0.31**	**6.77 ± 0.54**
	horse-ship	[27]	79.05 ± 31.04	46.90 ± 05.68
	horse-ship	**BetaPoisoning**	**06.41 ± 0.52**	**6.46 ± 0.38**
LC	frog-ship	[73]	16851.3 ± 2506.5	6386.32 ± 1076.1
	frog-ship	**BetaPoisoning**	**08.89 ± 0.48**	**08.82 ± 0.68**
	horse-ship	[73]	17788.9 ± 1335.1	7029.8 ± 343.08
	horse-ship	**BetaPoisoning**	**06.73 ± 0.71**	**06.58 ± 0.44**

variable reduction trick, we can optimize only 15 out of 784 variables for MNIST and 30 out of 3072 for CIFAR10. These results indicate that this approach is probably practicable on datasets with many more features than the CIFAR10 dataset.

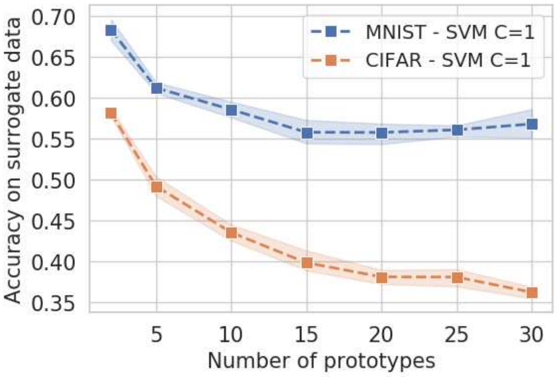


Fig. 4.7: *Effectiveness of poisoning when 15% of the dataset is poisoned. The x-axis represents the number of prototypes in $\mathcal{S}$, equal to the number of optimized variables. The y-axis shows the system's accuracy under attack on a validation/surrogate dataset.*

4.3 Concluding Remarks

In this Chapter, we examined the scalability of poisoning attacks, identifying a performance limitation, especially when running indiscriminate attacks. Specifically, we found that, unlike what has been developed for targeted or backdoor attacks, the literature on indiscriminate poisoning does not include any heuristic approaches to make them faster. The fundamental issue we identified is that they try to solve or approximate a complex bilevel optimization program. Nevertheless, we found that for simple scenarios, when the classifier is linear, we do not really need to solve the bilevel problem. Our analysis shows that, for this class of problems (the "nuts"), we do not need to use such a computational demanding bilevel formulation (the "hammer"). Indeed, we evince that our heuristic attack reaches comparable, or even better, results compared to theoretical and time-consuming formulations. We propose a re-parametrization trick to reduce the number of variables during the learning process. We compared the computational costs of the proposed algorithm with the ones obtained for "hammer-based" poisoning algorithms, namely the ones that solve exactly the bilevel optimization problem. We provided experimental evidence that we can poison target models with a significantly lower computational cost. Our approach may open the door to designing more efficient heuristics to deceive and test critical systems against indiscriminate data poisoning.

In conclusion, responding to our research question, "Can we make poisoning attacks scalable?", we would say "**yes, poisoning attacks can be scaled under specific circumstances, while they can not in more challenging scenarios**". Specifically, poisoning attacks are computationally demanding as they try to solve a complex bilevel problem (see also Sec. 3.3.4). Nevertheless, under specific circumstances (e.g., attacking the fine-tuning scenario, targeting linear classifiers, or having control of model training procedure), the attacker can exploit some heuristic approaches (e.g., feature collision [238], or our BetaPoisoning [60] attack, or fixed patch trigger [108]) to fasten them. However, their applicability in practice remains limited, and further investigation of more challenging scenarios (e.g., larger and more complex models, limited knowledge about the target system, data inspection, etc.) is required if the final goal is to have scalable attacks for malicious purposes or to build novel security benchmarks. In future works, we aim to extend our approach against non-linear classifiers to enlarge the applicability of our work.

Chapter 5

Understanding Backdoor Poisoning Vulnerability

Research Question #3

Which factors influence the effectiveness of backdoor poisoning attacks?

In this Chapter we respond to our RQ#3, which investigates the intriguing properties of ML models that make them vulnerable to backdoor threats and the attack capabilities that make them more effective. We believe that finding the factors influencing the model's vulnerability and attack effectiveness may stimulate the development of novel defensive or training techniques that preserve ML robustness or novel attacks for benchmarking them.

Underlying Problem. Despite the quickly-growing literature on identifying the causes of poisoning attacks, the majority of the previously proposed works study only targeted and indiscriminate poisoning attacks [96; 39; 260], not backdoors. Although backdoor poisoning attacks have been demonstrated in various settings and against different models, it is still unclear what factors influence a model's ability to learn a backdoor, i.e., to classify test samples containing the backdoor trigger as the class chosen by the attacker. Given the high applicability of this type of attack against real-world systems demonstrated in the literature, identifying the factors that influence the vulnerability of models to this threat is essential for developing future safeguards for users.

Related Work. Frederickson et al. [96] investigated on the trade-off between the attack strength, i.e., how much the attacker can influence the training set, and its detectability for indiscriminate poisoning attacks. Carnerero-Cano et al. [39] and Suya et al. [260] observed that regularization can be adopted for decreasing the effectiveness of indiscriminate and targeted poisoning attacks. Only a few works have studied factors that influence the success of backdoor poisoning. Baluta et al. [13] studied the relationship between backdoor effectiveness and the percentage of backdoored samples. Salem et al. [230] experimentally investigated the relationship between the backdoor effectiveness and the trigger size. We instead do not limit our study to neural networks but also study other models.
Furthermore, we also investigate the relationship between the model's complexity and backdoor effectiveness. Some of the defenses proposed against backdoors use different techniques to reduce complexity. These techniques include pruning [169; 12], data augmentation [307; 33] and gradient shaping [122]. However, from these works, it remains unclear why reducing complexity alleviates the threat of backdoor poisoning. To the best of our knowledge, our work is the first to investigate this aspect. We have analyzed the relationship between backdoor effectiveness and different factors, including complexity, controlled via the regularization and the RBF kernel's hyperparameter. Our

experimental results show that reducing complexity by choosing appropriate hyperparameter values improves robustness against backdoors.

Contributions and Outline. While for indiscriminate and targeted poisoning a few study exists on identifying the factors influencing their effectiveness, i.e., attacker's strength and model's regularization, we did not find a similar analysis for backdoor poisoning. Therefore, we investigated such direction in our work [59], explained in the remaining Chapter, to assess whether the factors found in previous work also influence the effectiveness of backdoor attacks or whether other aspects may regulate their performance. We thus introduce in Sec. 5.1 *backdoor learning curves* as a powerful tool to thoroughly characterize the backdoor learning process. We show in Sec. 5.1 that the slope of this curve, the *backdoor learning slope*, which is connected to the notion of influence functions [58; 144], quantifies the speed with which the model learns the backdoor samples and hence its vulnerability. Additionally, to provide further insights about the backdoor's influence on the learned classifiers, we propose in Sec. 5.1 a way to quantify the *backdoor impact on learning parameters*, i.e., how much the parameters of a model deviate from the initial values when the model learns a backdoor. Finally, we train different ML models under various hyperparameters and attack configurations, and we compare and analyze the obtained results in Sec. 5.2 to identify the factors influencing the backdoor performance. We conclude this Chapter in Sec. 5.3 by answering to our RQ#3, listing the factors influencing the backdoor effectiveness we identified thanks to our framework and we compare them with the ones affecting indiscriminate and targeted poisoning.

5.1 Backdoor Learning Process

In this section, we introduce our framework to characterize backdoor poisoning by means of learning curves and their slope. Afterwards, we introduce two measures to quantify the backdoor impact on the model's parameters.

Notation. We here recap the notation used in the remaining of this Chapter. Further details are reported in Chapter 3, and a list of symbols is given on page v.

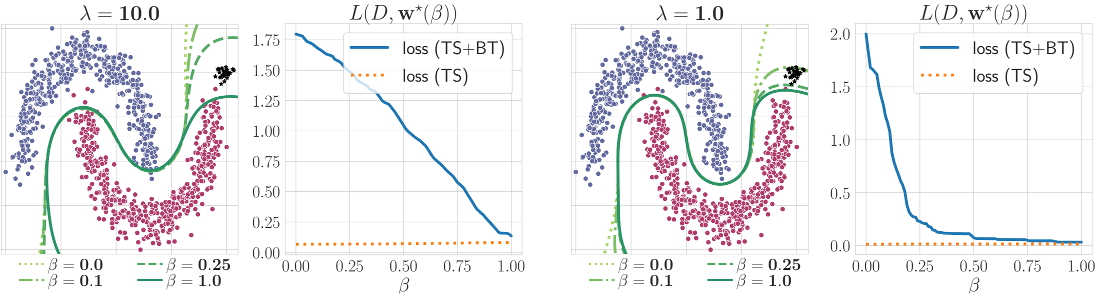
We denote the input data and their labels respectively with $\boldsymbol{x} \in \mathbb{R}^d$ and $y \in \{1, .., C\}$, being C the number of classes. We refer to the untainted, clean training data as $\mathcal{D} = (\boldsymbol{x}_i, y_i)_{i=1}^n$, and to the backdoor samples injected into the training set as $\mathcal{D}_p = (\hat{\boldsymbol{x}}_j, \hat{y}_j)_{j=1}^m$. We refer to the clean test samples as $\mathcal{T} = (\boldsymbol{x}_t, y_t)_{t=1}^k$ and to the test samples containing the backdoor trigger as $\mathcal{T}_p = (\hat{\boldsymbol{x}}_t, \hat{y}_t)_{t=1}^k$.

5.1.1 Backdoor Learning Curves

The learning process of humans is usually characterized using learning curves, a graphical representation of the relationship between the hours spent practicing and the proficiency to accomplish a given task. Inspired by this concept, we introduce the notion of *backdoor learning curves.* To generate these curves, we formulate backdoor learning as an incremental learning problem [41; 117] and assess how the loss on the backdoor samples decreases as they are gradually learned by the target model. In other words, with the backdoor learning curves we study how gradually incorporating backdoor samples affects the learned classifier. In mathematical terms, we formalize the learning problem as:

$$\boldsymbol{\theta}^\star(\beta) \in \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D} \cup \mathcal{D}_p, \boldsymbol{\theta}) = L(\mathcal{D}, \boldsymbol{\theta}) + \beta L(\mathcal{D}_p, \boldsymbol{\theta}) + \lambda \Omega(\boldsymbol{\theta}) \,, \tag{5.1}$$

where L is the loss attained on a given dataset by the classifier with parameters $\boldsymbol{\theta}$. $\mathcal{L}$ is the loss computed on the training points and the backdoor samples, which also includes a regularization term $\Omega(\boldsymbol{\theta})$ (e.g., $\|\boldsymbol{\theta}\|_2^2$), weighed by the regularization hyperparameter λ. To gradually incorporate the backdoor samples, we use $\beta \in [0, 1]$. It regulates the weight the classifier gives to the loss on the samples with backdoor trigger. As β ranges from 0 (unpoisoned classifier) to 1 (poisoned



(a) *Strong regularization ($\lambda = 10$).* **(b)** *Weak regularization ($\lambda = 1$).*

Fig. 5.1: *Backdoor learning curves and their relationship with model complexity. Considering an SVM with an RBF kernel ($\gamma = 10.0$) on a toy dataset in two dimensions, we show the influence of model complexity (controlled by λ, set to 10 (left) and 1 (right)). For each setting, we created two plots. The left plot shows on a 2-dimensional plane the data (dots) and decision surface for different values of β (green lines). The right plot shows the backdoor learning curve, where the y-axis represents the classifier's loss and the x-axis β. We plot both the loss on the clean test data (orange dotted line) and the test dataset containing the backdoor trigger (blue line). The slope of these curves represents the speed with which the model learns the backdoored samples (black dots), labeled as blue class. The model on the left struggles to fit the backdoors and succeeds only with high β, whereas the one on the right already fits the model at low β.*

classifier), the classifier gradually learns the backdoor by adjusting its parameters. For this reason, we make the dependency of the optimal weights $\boldsymbol{\theta}^\star$ on β explicit as $\boldsymbol{\theta}^\star(\beta)$.

We now define the *backdoor learning curve* as the curve showing the behavior of the classifier loss $L(\mathcal{T}_p, \boldsymbol{\theta}^\star(\beta))$ on the test samples with the backdoor trigger as a function of β. In the following, we abbreviate $L(\mathcal{T}_p, \boldsymbol{\theta}^\star(\beta))$ as L. Intuitively, the faster the backdoor learning curve decreases, the easier the target model is backdoored. The exact details of how the model is backdoored do not matter for this analysis, e.g. our approach captures for example both the setting where the training data is altered as well as the setting where fine-tuning data is tampered with.

We give an example of two such curves under different regularizations in Fig. 5.1. The left plots depict a strongly regularized classifier. The corresponding backdoor learning curve (on the right) shows that the classifier achieves low loss and high accuracy on the backdoor samples only after poisoning (when $\beta = 1$), namely when the loss on the backdoor samples is considered equally important to the loss on the training samples. The classifier on the right, instead, is less regularized and thus more complex. Consequently, this classifier learns to incorporate the backdoor samples much faster (at low β), namely when the loss on the backdoor points is taken into account less than the one on the training data. This highlights that this classifier is probably more vulnerable to this attack.

5.1.2 Backdoor Learning Slope

We quantify how fast an untainted classifier can be poisoned by proposing a novel measure, namely the *backdoor learning slope*, that measures the velocity with which the classifier learns to classify the backdoor samples correctly. This measure allows us to compare the vulnerability of a classifier trained with different hyperparameters or considering different poisoning scenarios (e.g. when the attacker can inject a different number of poisoning points or creating triggers with different sizes), allowing us to identify factors relevant to backdoor learning. Moreover, as we will show, this measure can be used by the system designer to choose an appropriate combination of hyperparameters for the task at hand. To this end, we define the *backdoor learning slope* as the gradient of the backdoor learning curve at $\beta = 0$, capturing the velocity of the curve on learning the backdoor. Formally, the *backdoor learning slope* can be formulated as follow:

$$\frac{\partial L(\mathcal{T}_p, \boldsymbol{\theta}^\star(\beta))}{\partial \beta} = \frac{\partial L}{\partial \boldsymbol{\theta}} \frac{\partial \boldsymbol{\theta}^\star}{\partial \beta}\,, \tag{5.2}$$

where the first term is straightforward to compute, and the second term implicitly captures the dependency of the optimal weights on the hyperparameter β. In other words, it requires us to understand how the optimal classifier parameters change when gradually increasing β from 0 to 1, i.e., while incorporating the backdoor samples into the learning process.

To compute this term, as suggested in previous work in incremental learning [41], we assume that, while increasing β, the solution maintains the optimality (Karush-Kuhn-Tucker, KKT) conditions intact. This equilibrium implies that $\nabla_\beta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}^\star) + \frac{\partial \boldsymbol{\theta}^\star}{\partial \beta} \nabla^2_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}^\star) = \mathbf{0}$. Based on this condition, we obtain the derivative of interest,

$$\frac{\partial \boldsymbol{\theta}^\star}{\partial \beta} = -(\nabla^2_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}^\star))^{-1} \cdot \nabla_\beta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}^\star)\,. \tag{5.3}$$

Substituting it in Eq. (5.2) we obtain the complete gradient:

$$\frac{\partial L(\mathcal{T}_p, \boldsymbol{\theta}^\star(\beta))}{\partial \beta} = -\nabla_{\boldsymbol{\theta}} L \cdot (\nabla^2_{\boldsymbol{\theta}} \mathcal{L})^{-1} \cdot \nabla_\beta \nabla_{\boldsymbol{\theta}} \mathcal{L}\,. \tag{5.4}$$

The gradient in Eq. (5.4) corresponds to the sum of the pairwise influence function values $\mathcal{I}_{\mathrm{up,loss}}(\boldsymbol{x}_{\mathrm{tr}}, \boldsymbol{x}_{\mathrm{ts}})$ used by [144]. The authors indeed proposed to measure how relevant a training point is for the predictions of a given test point by computing $\frac{\partial L}{\partial \beta}\Big|_{\beta=0} = \sum_t \sum_j \mathcal{I}_{\mathrm{up,loss}}(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{x}}_j)$. In our work, we thus clarify that influence functions naturally descend from the incremental learning formulation in Eq. (5.1), showing that they quantify the velocity with which the classifier will learn new points.

To understand how the gradient in Eq. (5.4) can be efficiently computed via Hessian-vector products and other approximations, we refer the reader to [144] as well as to recent developments in gradient-based bilevel optimization [178; 95; 215; 184; 77]. Moreover, we show in Sec. 5.2 (Fig. 5.13-5.14) an example of the usage of influence functions for weakly and strongly regularized models.

The main difference between the approach by [144] and ours stems from their implicit treatment of regularization and our interest in understanding vulnerability to a subset of backdoor training points, rather than in providing prototype-based explanations. However, directly using the gradient of the loss wrt. β comes with two disadvantages. First, the slope is inverse to β and second, to obtain results comparable across classifiers, we need to rescale the slope. We thus transform the gradient as:

$$\eta = -\frac{2}{\pi} \arctan\left(\frac{\partial L}{\partial \beta}\Big|_{\beta=0} \right) \in [-1, 1]\,, \tag{5.5}$$

where we use the negative sign to have positive values correlated with faster backdoor learning (i.e., the loss decreases faster as β grows). Computing $2/\pi$ of the gradient allows us to rescale the slope to be in the interval between $[-1, 1]$. Hence, a value around 0 implies that the loss of the backdoor samples does not decrease, i.e., the classifier is robust enough that it does not learn the backdoor trigger.

5.1.3 Backdoor Impact on Learning Parameters

After introducing the previous plot and measure, we are able to quantify how backdoors are learned by the model. To provide further insights about the backdoor's influence on the learned classifier, we propose to monitor how the classifier's parameters deviate from their initial, unbackdoored values once a backdoor is added. Our approach below captures only convex learners. As shown by Zhang et al. [309], the impact of a network weight in non-convex classifiers' decision depends on the layer of which it is part. Therefore, measuring the parameter deviation in the non-convex case is challenging, and we leave this unsolved problem for future work.

To capture backdoor impact on learning parameters in the convex case, we consider the initial weights $\boldsymbol{\theta}_0 = \boldsymbol{\theta}^\star(\beta = 0)$ and $\boldsymbol{\theta}_\beta = \boldsymbol{\theta}^\star(\beta)$ for $\beta > 0$, and measure two quantities:

$$\rho = \|\boldsymbol{\theta}_\beta\| \in [0, \infty), \text{ and } \nu = \frac{1}{2}\left(1 - \frac{\boldsymbol{\theta}_0^\top \boldsymbol{\theta}_\beta}{\|\boldsymbol{\theta}_0\|\|\boldsymbol{\theta}_\beta\|}\right) \in [0, 1]\,. \tag{5.6}$$

The first measure, ρ, quantifies the change of the weights when β increases. This quantity is equivalent to the regularization term used for learning. The second one, ν, quantifies the change in orientation of the classifier. In a nutshell, we compute the angle between the two vectors and rescale it to be in the interval of $[0, 1]$. Both metrics are defined to grow as $\beta \to 1$, in other words as the backdoored classifier deviates more and more from the original classifier. Consequently, in the empirical parameter deviation plots in Sec. 5.2, we report the value of $\rho(\nu)$ (on the y-axis) as β (on the x-axis) varies from 0 to 1, to show how the classifier parameters are affected by backdoor learning.

5.2 Experimental Analysis

Employing the previously proposed methodology, we carried out an empirical analysis on linear and nonlinear classifiers. In this section, we start with the experiments aimed to study the impact of different factors on backdoor learning. To this end, we employ the backdoor learning curves and the backdoor learning slope to study how the capacity of the model to learn backdoors changes when (a) varying the model's complexity, defined by its hyperparameters, (b) the attacker's strength, defined by the percentage of poisoning samples in the training set and (c) the trigger size and visibility. Our results show that these components significantly determine how fast the backdoor is learned, and consequently, the model's vulnerability. Then, leveraging the proposed measures to analyze how the classifier's parameters change during backdoor learning, we provide further insights on the effect of the aforementioned factors on the trained model. The results presented in this section will help identify novel criteria to improve existing defenses and inspire new countermeasures. The source code is available at `github.com/Cinofix/backdoor_learning_curves`.

5.2.1 Experimental Setup

Our work investigates which factors influence backdoor vulnerability considering convex-learners and neural networks. In the following, we describe our datasets, models, and the backdoor attacks studied in our experiments.

Datasets. We carried out our experiments on the MNIST [157], CIFAR10 [147] (already seen in Sec. 4.2) and Imagenette [126]. Imagenette is a subset of 10 classes from Imagenet. We use the 320px version, where the shortest side of each image is resized to that size.
When training convex-learners we consider the two-class subproblems as in the work by Saha et al. [229] and Suya et al. [260]. On MNIST, we choose the pairs 7 vs 1, 3 vs 0, and 5 vs 2, as our models exhibited the highest clean accuracy on these pairs. On CIFAR10, analogous to prior work [229], we choose *airplane vs frog*, *bird vs dog*, and *airplane vs truck*. On Imagenette we randomly choose *tench vs truck*, *cassette player vs church*, and *tench vs parachute*. For each two-class subtask we use 1500 and 500 samples as training and test set respectively. In the following section, we focus on the results of one pair on each dataset: 7 vs 1 on MNIST, *airplane vs frog* on CIFAR10, and *tench vs truck* on Imagenette. The results of the other pairs (reported in A) are analogous. When testing our framework against neural networks, we train on all the ten classes of Imagenette. We use 70% and 30% of the entire dataset for training and test, respectively.

Models and Training phase. To thoroughly analyze how learning a backdoor affects a model, we consider different convex learning algorithms, including linear Support Vector Machines (SVMs), Logistic Regression Classifiers (LCs), Ridge Classifiers (RCs) and nonlinear SVMs using the Radial

Basis Function (RBF) kernel, and deep neural networks. We train the classifiers directly on the pixel values scaled in the range $[0, 1]$ on the MNIST dataset. For CIFAR10 and Imagenette, we instead consider a transfer learning setting frequently adopted in the literature [78; 213; 144]. Like Saha et al. [229], we use the pre-trained model AlexNet [148] as a feature extractor. The convex-learners are then trained on top of the feature extractor. We study these convex learners due to their broad usage in industry [260], derived from their low computational cost, excellent results, and good interpretability [267; 69].
Regarding the considered deep neural networks, we use pretrained Resnet18 and Resnet50[1] which are among the most widely used architectures [309]. We fine-tune them on the Imagenette dataset.

Hyperparameters. The choice of hyperparameters has a relevant impact on the learned decision function. For example, some of these parameters control the complexity of the learned function, which may lead to overfitting [190], thereby potentially compromising classification accuracy on test samples. We argue that a high complexity may also lead to higher importance to outlying samples, including backdoors, and thus has a crucial impact on the capacity of the model to learn backdoors. To verify our hypothesis, we consider different configurations of the models' hyperparameters. For convex-learners we consider two hyperparameters that impact model complexity, i.e., the regularization hyperparameter λ and the RBF kernel hyperparameter γ. To this end, we take 10 values for λ on a uniformly spaced interval on a log scale from 1e−04 to 1e+02. For the Imagenette dataset we extend this interval in $[1\text{e}{-05}, 1\text{e}{+02}]$. Concerning the RBF kernel, we let γ take 5 uniformly spaced values on a log scale in $[5\text{e}{-04}, 5\text{e}{-02}]$ for MNIST, $[1\text{e}{-04}, 1\text{e}{-02}]$ for CIFAR, and $[1\text{e}{-05}, 1\text{e}{-03}]$ for Imagenette. Furthermore, we take 10 values of λ in the log scale uniformly spaced interval $[1\text{e}{-01}, 1\text{e}{+02}]$ for the RBF kernel. This allowed us to study a combination of 10 and 50 hyperparameters for linear classifiers and RBF SVM respectively.
For deep neural networks, we consider two different numbers of epochs: 10 and 50, and increase the number of neurons when using Resnet50 instead of Resnet18. Whereas size intuitively correlates with complexity, previous works, including [40], show that decreasing the number of training epochs reduces the complexity of the trained network as well. Conversely, increasing epochs leads to overfitting on the training dataset, thus a more complex decision function. Each network is fine-tuned using the SGD optimizer with a learning rate of 0.001, a momentum of 0.9, and batch size 256.

Backdoor Attacks. We implement the backdoor attacks proposed by Gu et al. [108] against MNIST and CIFAR10. More concretely, we use a random 3×3 patch as the trigger for MNIST, while on CIFAR10, we increase the size to 8×8 to strengthen the attack [229]. We add the trigger pattern in the lower right corner of the image [108]. Samples from MNIST and CIFAR10 with and without trigger can be found in Fig. 5.12. However, in contrast to previous approaches [108], we use a separate trigger for each base-class i. The reason is that our study encompasses linear models that are unable to associate the same trigger pattern to two different classes. Using different trigger patterns, we enhance the effectiveness of the attack on these linear models. On the Imagenette dataset, we use the backdoor trigger developed by [322]. This attack consists of injecting a patterned perturbation mask into training samples to open the backdoor. A constant value c_m refers to the maximum allowed intensity. We apply the backdoor attacks by altering 10% of the training data if not stated otherwise, and, as done by Gu et al. [108], we force the backdoored model to predict the i-th class as class $(i + 1)\% n\_classes$ when the trigger is shown. We also report additional experiments concerning variations in the trigger's size or visibility.

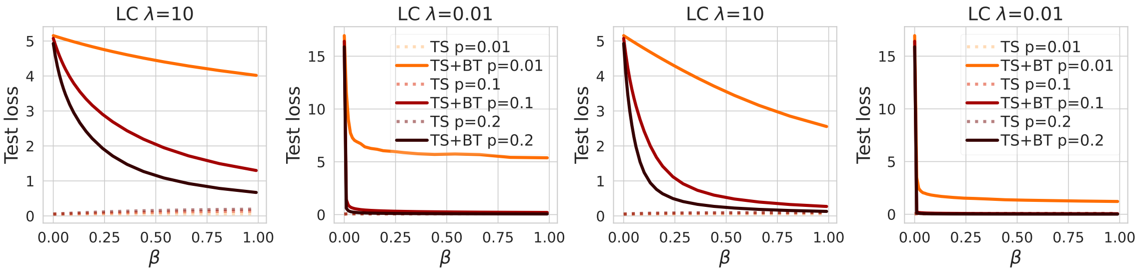
5.2.2 Experimental Results

In the following we now discuss our experimental results obtained with the datasets, classifiers

---

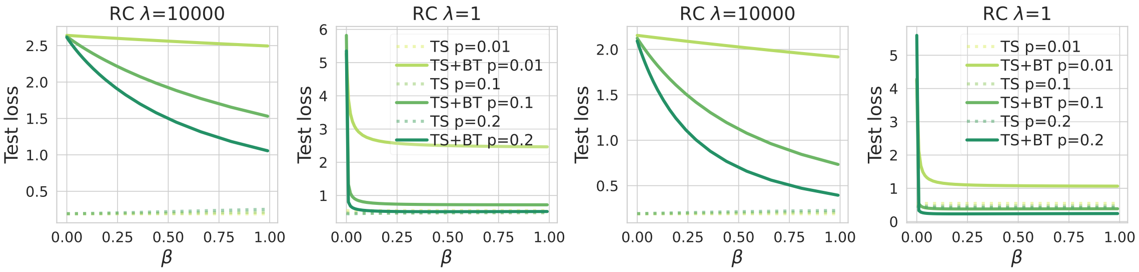
[1] From the torchvision repository `https://pytorch.org/vision/stable/models.html`.

and backdoor attacks described above.



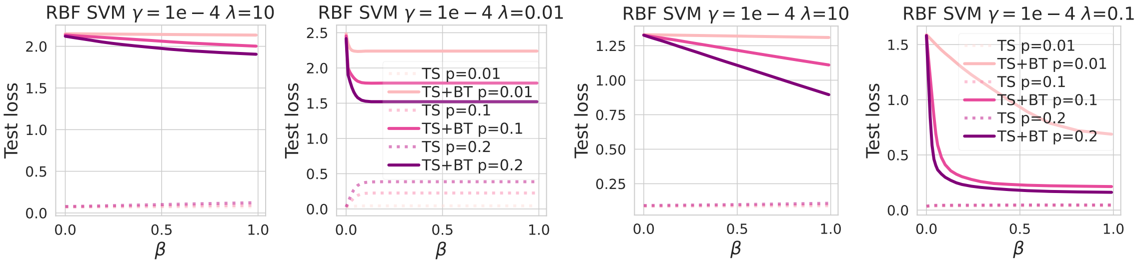
(a) *MNIST trigger size* 3×3.

(b) *MNIST trigger size* 6×6.



(c) *CIFAR10 trigger size* 8×8.

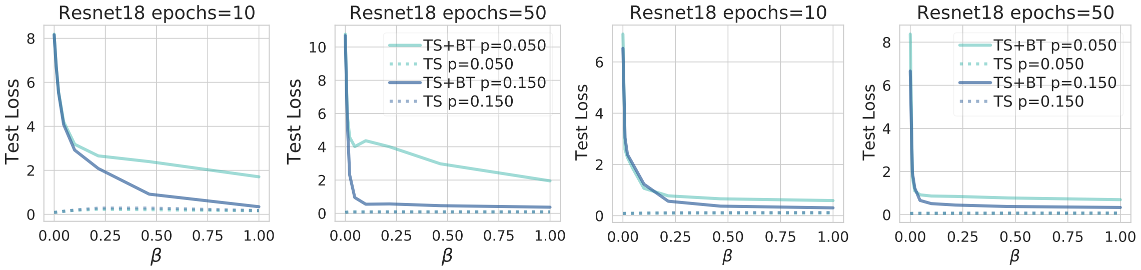
(d) *CIFAR10 trigger size* 16×16.



(e) *Imagenette trigger visibility* $c_m = 10$.

(f) *Imagenette trigger visibility* $c_m = 75$.

Fig. 5.2: *Backdoor learning curves for: (top row) logistic classifier (LC) on MNIST 7 vs. 1 with* $\lambda \in \{10, 0.01\}$ *and trigger size* 3×3 *(left) or* 6×6 *(right); (middle row) Ridge classifier on CIFAR10 airplane vs frog with* $\lambda \in \{100000, 100\}$ *and trigger size* 8×8 *(left) or* 16×16 *(right); (bottom row) RBF SVM with* $\gamma = 1\text{e}{-04}$ *on Imagenette tench vs truck with* $\lambda \in \{10, 0.1\}$ *and trigger visibility* $c_m = 10$ *(left) or* $c_m = 75$ *(right). Darker lines represent a higher fraction of poisoning samples p injected into the training set. We report the loss on the clean test samples (TS) with a dashed line and on the test samples with the backdoor trigger (TS+BT) with a solid line.*



(a) *Trigger visibility* $c_m = 10$. **(b)** *Trigger visibility* $c_m = 75$.

Fig. 5.3: *Backdoor learning curves for Resnet18 trained on the full Imagenette training dataset with* 10 *and* 50 *epochs. Darker lines represent a higher fraction of poisoning samples* p *injected into the training set. We report the loss on the clean test samples (TS) with a dashed line and on the test samples with the backdoor trigger (TS+BT) with a solid line.*

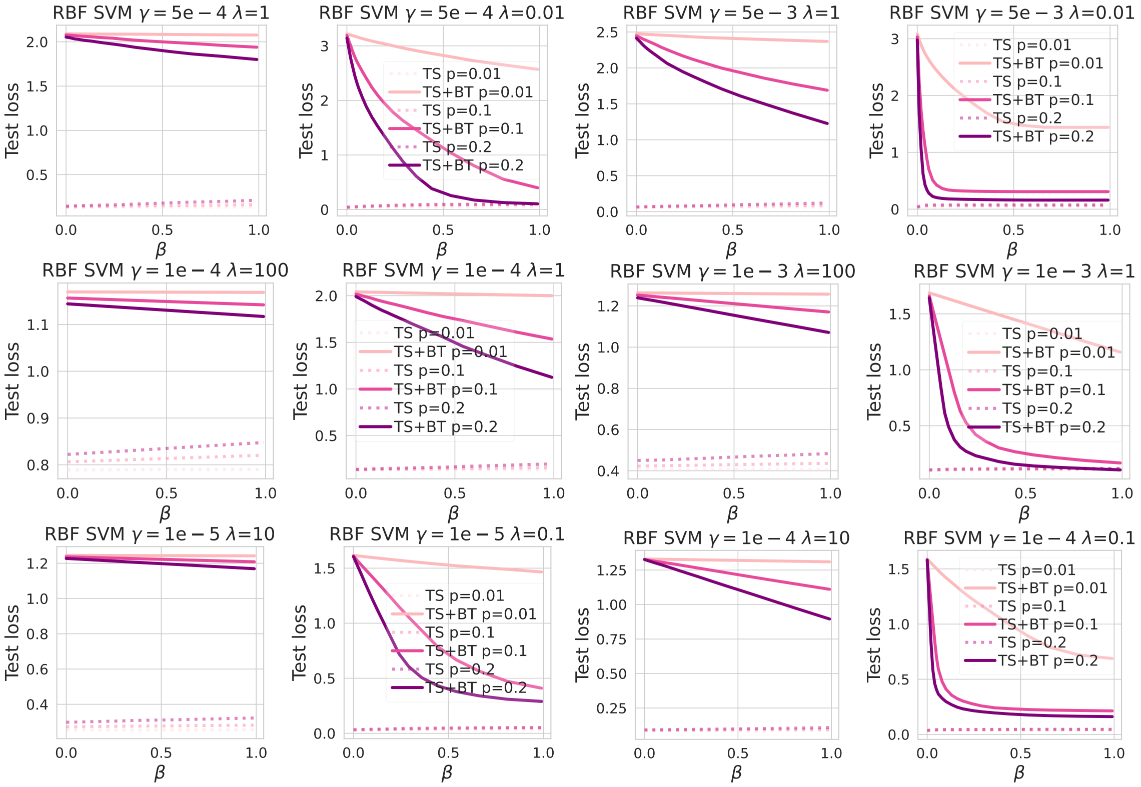


Fig. 5.4: *Backdoor learning curves for MNIST* 7 *vs* 1 *(top row), CIFAR10 airplane vs frog (middle row) and Imagenette tench vs truck (bottom row) when changing the kernel parameter* γ *on RBF SVM. Darker lines represent a higher fraction of poisoning samples* p *injected into the training set. We report the loss on the clean test samples (TS) with a dashed line and on the test samples with the backdoor trigger (TS+BT) with a solid line.*

Backdoor Learning Curves. Here we present the results obtained using the learning curves that we proposed to study the impact of three different factors on the backdoor learning process: (i) model complexity, (ii) the fraction of backdoor samples injected, and (iii) the size and visibility of the backdoor trigger. We report the impact of the these factors on the backdoor learning curves in Fig. 5.2 and 5.3.
More specifically, in Fig. 5.2 we consider convex classifiers (i.e. LC, RC and RBF SVM) trained on two-class subproblems (MNIST, CIFAR10 and Imagenette), whereas in Fig. 5.3 we show the results for Resnet18 trained on all the ten classes of Imagenette. To analyze the first factor, we report the results on the same classifiers, changing the hyperparameters that influence their corresponding complexity. In the case of convex-learners, we test different values of the regularization coefficient, while for Resnet18, we increase the number of epochs.

To analyze the impact of the second factor, we plot the backdoor learning curves when the attacker injects an increasing percentage of poisoning points $p \in \{0.01, 0.1, 0.2\}$ for convex learners and $p \in \{0.05, 0.15\}$ for Resnet18. Finally, to study the third factor, namely the size and visibility of the backdoor trigger, we have created the same backdoor curves doubling the size of the patch triggers for MNIST and CIFAR10, and increasing the trigger's visibility for Imagenette. Even when a high percentage of poisoning points are injected, for flexible enough classifiers, the loss on the clean test samples remains almost constant. Instead, the loss on the test set containing the backdoor trigger is highly affected by the factors mentioned above. Both a smaller λ or a larger number of epochs (low regularization and thus higher complexity), and larger p (a high percentage of poisoning points added) increase the slope of the backdoor learning curve. This means that the classifier learns the backdoor faster. When the classifier is sufficiently complex, even a low percentage of low poisoning points suffices to make the classifier learn the backdoor rapidly. On the other hand, this does not hold for highly regularized classifiers, which generally learn backdoors slowly. Therefore, limiting the classifier complexity by choosing an appropriate regularization coefficient may reduce the vulnerability against backdoors.

Moreover, our results show that if the trigger size is large, the classifiers learn the backdoor faster, especially if they are complex. The same conclusion holds when increasing the trigger's visibility, thus shedding light on the familiar trade-off between attacker's strength vs. detectability introduced by Frederickson et al. [96]. The attacker can increase the trigger size or increase the trigger's visibility to make the backdoor more effective. However, at the same time, these changes enable the defender to detect the attack more easily.

Concerning the RBF SVM's robustness to backdoors, we analyzed the backdoors' learning curves for different values of γ which determine the RBF kernel's curvature. More precisely, depending on γ, we have analyzed the backdoor learning curves, and the classifier's parameters change due to backdoor learning. We plot the learning curves in Fig. 5.4. On both datasets, decreasing γ leads to flatter backdoor learning curves and increased test loss, suggesting higher robustness.

Overall, our experiments show that to learn a backdoor, a classifier has to increase its complexity (if it is not already highly complex). Such an increase in complexity is limited when the classifier is highly regularized. Such regularized classifiers are thus, in terms of backdoor robustness, preferable. We show the same plots for other classifiers in A, which confirm the trends we highlight here.

Backdoor Slope. From the previous results we have seen that reducing complexity through regularization increases robustness against backdoors. For a deeper understanding of model complexity on backdoor learning, we leverage the proposed backdoor slope. In our experiments for convex-learners, we fix the fraction of injected poisoning points equal to 0.1, as by Gu et al [108], and we report a dot for each combination of λ and γ as specified in Sec. 5.2. Fig. 5.5-5.7 show the relationship between the backdoor slope and the backdoor effectiveness, measured as the percentage of samples with trigger that mislead the classifier, respectively for MNIST, CIFAR10 and Imagenette. We report the accuracy on the clean test dataset and on the test dataset with the backdoor trigger. For the SVM with RBF kernel, we report the accuracies for two different gamma.

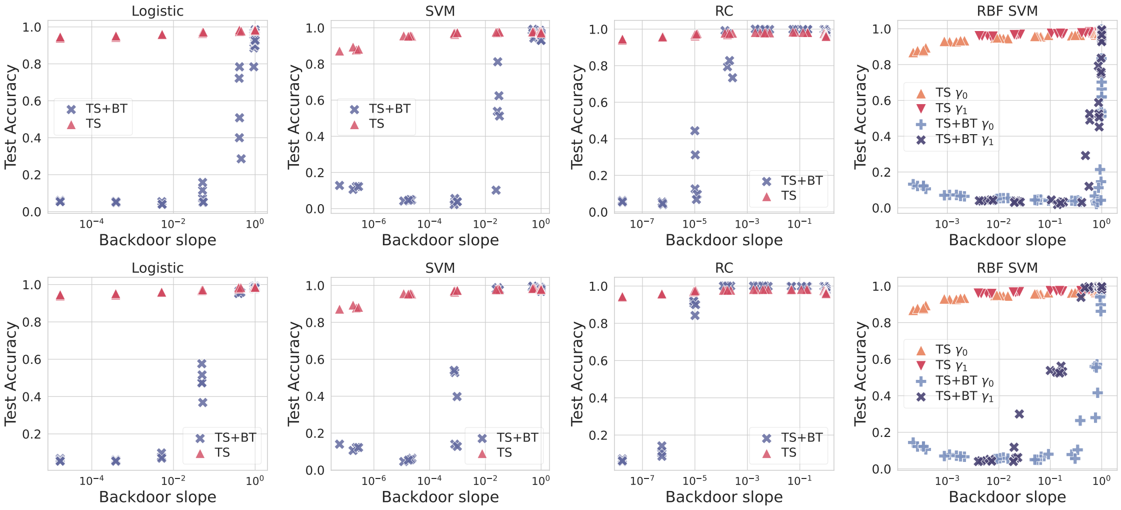


Fig. 5.5: *Backdoor slope* η *vs clean accuracy (red) and backdoor effectiveness (blue) on MNIST 7 vs. 1 with backdoor trigger size* 3×3 *(top row) and* 6×6 *(bottom row). We measure the classification accuracy on the untainted test samples (TS), and on the same samples after adding the backdoor trigger (TS+BT). We chose the* γ *parameter for the RBF kernel as* $\gamma_0 = 5\text{e}{-04}$ *(orange triangle for clean data, light blue plus for data with trigger) and* $\gamma_1 = 5\text{e}{-03}$ *(red inverted triangle for clean data, dark blue x for data with trigger).*

Interestingly, our plots show a region where the accuracy of the classifier on benign samples is high, yet the classifier exhibits low accuracy on samples with trigger. For linear classifiers, this region equals low-regularized classifiers. In case of the RBF SVM, the best trade-off is achieved with high λ (strong regularization) and small γ, which also constrain SVM's complexity. Our results thus indicate that in these cases, the classifier is not flexible enough to learn the backdoor in addition to the clean test samples. Conversely, as long as the classifier has enough flexibility, it is able to learn the backdoor without sacrificing clean test accuracy. In a nutshell, choosing the hyperparameters appropriately, we can obtain a classifier able to learn the original task but not the backdoor. However, there is a trade-off between the accuracy on the original task and robustness to backdoor classification. In Fig. 5.5-5.7 we further compare the relationship between backdoor learning slope and backdoor effectiveness considering a stronger attack (larger or more visible trigger). For these attacks, the trade-off region is reduced, leaving fewer possible configurations of the hyperparameters that yield a robust model. This result is consistent with our previous results using backdoor learning curves: the faster the curve descends when the trigger strength is increased, the higher the backdoor slope. Our results suggest that system designers should regularize models as much as possible while accepting a small accuracy loss in order to deploy a trustworthy ML algorithm.

As a final check to asses the reliability of the backdoor learning slope, we plot in Fig. 5.8 clean and backdoor accuracy when training an RBF SVM with different hyperparameter configurations. We train, for each configuration, the classifier on the poisoned dataset. On the top row we show the results for CIFAR10 *airplane vs frog*, and on bottom row the results for Imagenette *tench vs truck*. We followed the same backdoor setting for the backdoor slope in Fig. 5.5-5.7, i.e. trigger size 8×8 for MNIST and trigger visibility $c_m = 75$ for Imagenette. Analogous to our previous findings, there exists a trade-off region where the clean accuracy is high (red), while the backdoor accuracy is low (blue), suggesting an higher robustness against backdoors. Analogous to the backdoor slope, the best region is obtained with reduced complexity, thus regularizing it or reducing γ. Although these matrices yield the same conclusion, it is worth to remark that computing the backdoor learning slope does not require to re-train the a specific classifier for each configuration on the poisoned training dataset. Therefore, backdoor learning slope can be used

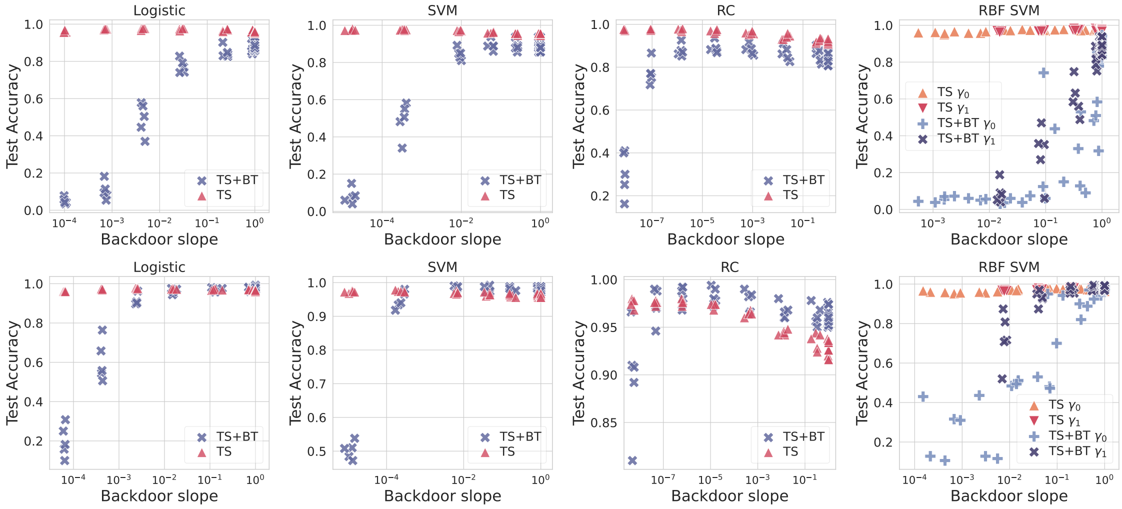


Fig. 5.6: *Backdoor slope η vs clean accuracy (red) and backdoor effectiveness (blue) on CIFAR10 airplane vs frog with backdoor trigger size 8×8 (top row), and 16×16 (bottom row). We measure the classification accuracy on the untainted test samples (TS), and on the same samples after adding the backdoor trigger (TS+BT). We chose the γ parameter for the RBF kernel as $\gamma_0 = 1\text{e}{-04}$ (orange triangle for clean data, light blue plus for data with trigger) and $\gamma_1 = 1\text{e}{-03}$ (red inverted triangle for clean data, dark blue x for data with trigger).*

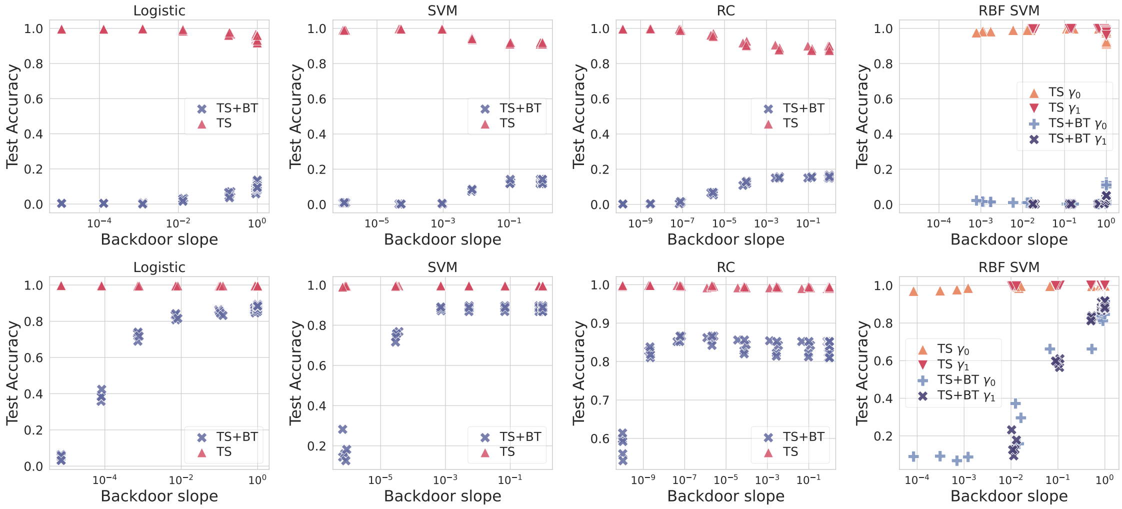
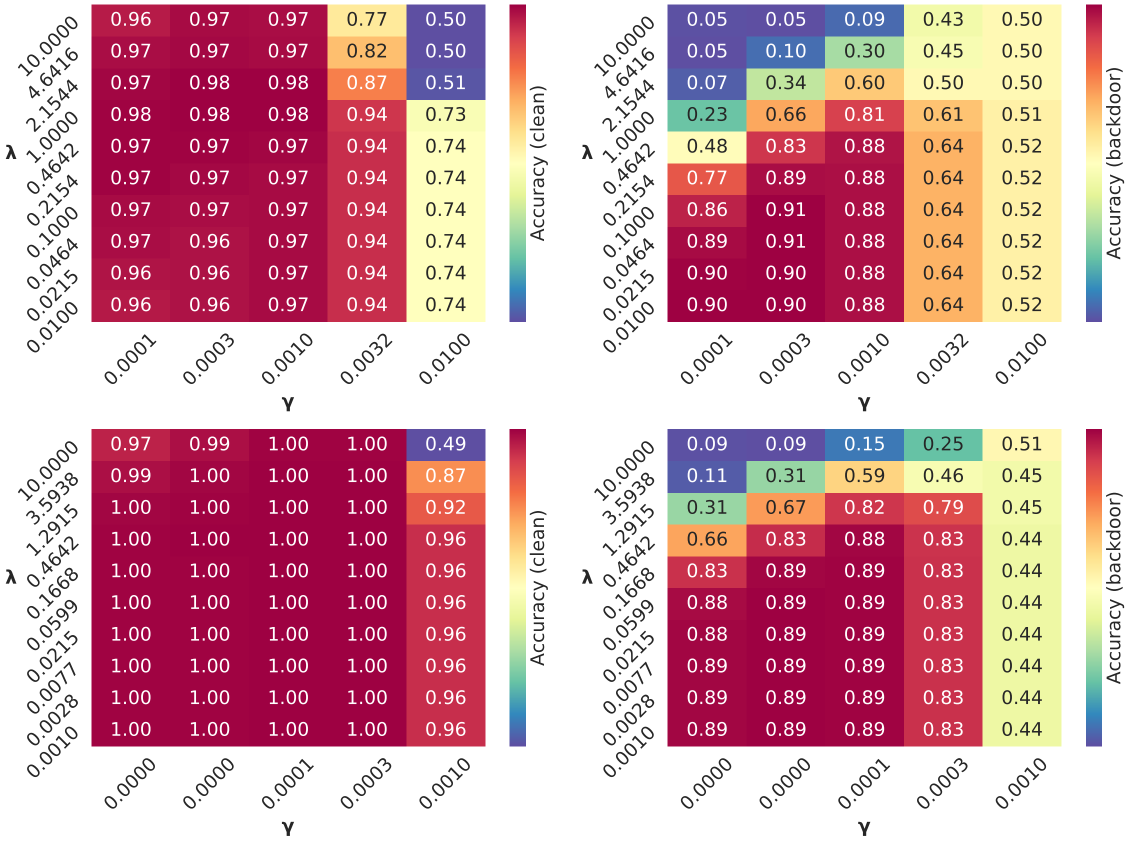


Fig. 5.7: *Backdoor slope η vs clean accuracy (red) and backdoor effectiveness (blue) on Imagenette tench vs truck with trigger visibility $c_m = 10$, i.e. almost imperceptible, (top row) and $c_m = 75$ (bottom row). We measure the classification accuracy on the untainted test samples (TS), and on the same samples after adding the backdoor trigger (TS+BT). We chose the γ parameter for the RBF kernel as $\gamma_0 = 1\text{e}{-05}$ (orange triangle for clean data, light blue plus for data with trigger) and $\gamma_1 = 1\text{e}{-04}$ (red inverted triangle for clean data, dark blue x for data with trigger).*

as tool for improving and speeding-up the hyperparameter optimization procedure, to look for accurate yet robust models.

While this measure works well on convex learners, its roots in influence functions prevent a direct application on neural networks. As pointed out in [18] the analytical gradient in Eq. (5.5) at $\beta = 0$ is unstable for DNNs. To overcome this deficiency we estimate it with finite difference



λ	0.0001	0.0003	0.0010	0.0032	0.0100
10.0000	0.96	0.97	0.97	0.77	0.50
4.6416	0.97	0.97	0.97	0.82	0.50
2.1544	0.97	0.98	0.98	0.87	0.51
1.0000	0.98	0.98	0.98	0.94	0.73
0.4642	0.97	0.97	0.97	0.94	0.74
0.2154	0.97	0.97	0.97	0.94	0.74
0.1000	0.97	0.97	0.97	0.94	0.74
0.0464	0.97	0.96	0.97	0.94	0.74
0.0215	0.96	0.96	0.97	0.94	0.74
0.0100	0.96	0.96	0.97	0.94	0.74

λ	0.0001	0.0003	0.0010	0.0032	0.0100
10.0000	0.05	0.05	0.09	0.43	0.50
4.6416	0.05	0.10	0.30	0.45	0.50
2.1544	0.07	0.34	0.60	0.50	0.50
1.0000	0.23	0.66	0.81	0.61	0.51
0.4642	0.48	0.83	0.88	0.64	0.52
0.2154	0.77	0.89	0.88	0.64	0.52
0.1000	0.86	0.91	0.88	0.64	0.52
0.0464	0.89	0.91	0.88	0.64	0.52
0.0215	0.90	0.90	0.88	0.64	0.52
0.0100	0.90	0.90	0.88	0.64	0.52

λ	0.0000	0.0000	0.0001	0.0003	0.0010
10.0000	0.97	0.99	1.00	1.00	0.49
3.5938	0.99	1.00	1.00	1.00	0.87
1.2915	1.00	1.00	1.00	1.00	0.92
0.4642	1.00	1.00	1.00	1.00	0.96
0.1668	1.00	1.00	1.00	1.00	0.96
0.0599	1.00	1.00	1.00	1.00	0.96
0.0215	1.00	1.00	1.00	1.00	0.96
0.0077	1.00	1.00	1.00	1.00	0.96
0.0028	1.00	1.00	1.00	1.00	0.96
0.0010	1.00	1.00	1.00	1.00	0.96

λ	0.0000	0.0000	0.0001	0.0003	0.0010
10.0000	0.09	0.09	0.15	0.25	0.51
3.5938	0.11	0.31	0.59	0.46	0.45
1.2915	0.31	0.67	0.82	0.79	0.45
0.4642	0.66	0.83	0.88	0.83	0.44
0.1668	0.83	0.89	0.89	0.83	0.44
0.0599	0.88	0.89	0.89	0.83	0.44
0.0215	0.88	0.89	0.89	0.83	0.44
0.0077	0.89	0.89	0.89	0.83	0.44
0.0028	0.89	0.89	0.89	0.83	0.44
0.0010	0.89	0.89	0.89	0.83	0.44

Fig. 5.8: *Influence of γ (x-axis) and λ (y-axis) on the backdoor effectiveness (left) and clean accuracy (right) for CIFAR10 airplane vs frog (top row) and Imagenette tench vs truck (bottom row). Backdoor is mounted with trigger size 8×8 for CIFAR10 and visibility $c_m = 75$ for Imagenette. The plots show that there are hyperparameter configurations for which clean accuracy is high (red regions on the left plots), while the accuracy on the backdoored points is low (blue regions on the right plots).*

approximation, obtaining:

$$\left.\frac{\partial L}{\partial \beta}\right|_{\beta=0} = \frac{L(\mathcal{T}_p, \boldsymbol{\theta}^\star(h)) - L(\mathcal{T}_p, \boldsymbol{\theta}^\star(0))}{h}\,, \tag{5.7}$$

We report the the results for Resnet18 and Resnet50 in Table 5.1 where we used $h = \{0.01, 0.1, 0.2, 1\}$. For each combination of poisoning percentage and number of epochs, we report the estimate of the backdoor learning slope when choosing different h values. The closer h is to 0, the closer to 1 is the backdoor slope of the neural network. This result is consistent with Fig. 5.3, where the backdoor learning curves drop similarly fast, suggesting a high vulnerability of the model in the presence of backdoor samples. A subtle difference is that when increasing h, there is more evidence for higher vulnerability of neural networks trained with more epochs or when increasing the percentage of poisoning points.

In conclusion, after seen the results for convex-learners and neural networks, we state that a wise choice of the hyperaparameters, such as the regularization term λ, number of epochs or neurons, can allow the user to find a good tradeoff between accuracy on benign samples and robustness to backdoor poisoning.

Empirical Parameter Deviation Plots. After having investigated which factors influence backdoor effectiveness, we now focus our attention on how the model adapts/changes its parameters

Table 5.1: *Backdoor learning slope for Resnet18 and Resnet50 when increasing the percentage of backdoor poisoning p, the number of epochs (#Epochs), and parameter h for estimate in Eq.* (5.7). *We also report the corresponding backdoor effectiveness (Accuracy TS+BT) and clean accuracy (Accuracy TS+BT), measures respectively as the percentage correctly classified test samples with and without the backdoor trigger.*

Model	p	#Epochs	Slope h=0.01	Slope h=0.1	Slope h=0.2	Slope h=1	Accuracy TS+BT	Accuracy TS
Resnet18	0.05	10	0.9955	0.9872	0.9752	0.9026	0.4163	0.9588
Resnet50	0.05	10	0.9965	0.9895	0.9785	0.9169	0.7197	0.9781
Resnet18	0.05	50	0.9986	0.9900	0.9797	0.9281	0.5256	0.9737
Resnet50	0.05	50	0.9992	0.9936	0.9849	0.9377	0.8067	0.9881
Resnet18	0.15	10	0.9955	0.9878	0.9774	0.9189	0.8804	0.9568
Resnet50	0.15	10	0.9966	0.9902	0.9943	0.9231	0.9440	0.9826
Resnet18	0.15	50	0.9987	0.9937	0.9864	0.9384	0.8893	0.9720
Resnet50	0.15	50	0.9992	0.9939	0.9971	0.9403	0.9509	0.9890

during the backdoor learning process, whether there is an increase in complexity or not. We use our two measures proposed in Sec. 5.1, ρ and ν to analyze the parameter change. The former, ρ, monitors the change of the weights, for example whether they increase or decrease. The latter, ν, measures the change in orientation or angle of the classifier. We plot both measures with different regularization parameters, trigger size or visibility with fraction of poisoning points to $p = 0.1$ in Fig. 5.9-5.11.

On linear classifiers, $\rho(\boldsymbol{\theta})$ increases during the backdoor learning process. This equals an increase of the weights' values, suggesting that the classifiers become more complex while learning the backdoor. However, when investigating the RBF SVM, the results are slightly different. Indeed, when increasing γ and decreasing λ, the classifier becomes flexible and complex enough to learn the backdoor without increasing its complexity. On the other hand, when decreasing γ, the model is constrained to behave similarly to a linear classifier. In this way, analogously to linear classifiers, the model needs to increase its complexity to learn the backdoor. When increasing the trigger size or visibility the results are similar, thus confirming the previous analysis. However, as a results of increasing the attacker's strength, the backdoor accuracy turns out to be higher.

Explaining Backdoor Predictions. In the following, we give a graphical interpretation of the poisoned convex-classifier's decision function, expressed by its internal weights, for which interpretation of their results is easier [69; 267]. We consider the results for backdoor trigger [108] in a specific position, as its influence on the classifier decision is easier to see. Conversely, the backdoor trigger by for example Zhong et al. [322] spans to the entire image, and therefore its influence is harder to spot from the interpretability plots. In particular, given a sample x we aim to compute and show the gradient of the classifier's decision function with respect to x. We use an SVM with regularization $\lambda = 1\text{e}{-02}$ for MNIST 7 vs 1 and CIFAR10 *airplane vs frog*, and report the results in Fig. 5.12.

For MNIST we consider a digit 7 with the trigger (left) and we show the gradient of the clean classifier's decision function. We report the outcome of the gradient from the clean (middle) and poisoned (right) classifiers for the corresponding input. Since we train a linear classifier on the input space, the derivative coincide with the classifier's weights. Intriguingly, the classifier's weights increased their magnitude and now have high values on the bottom right corner, where the trigger is located.

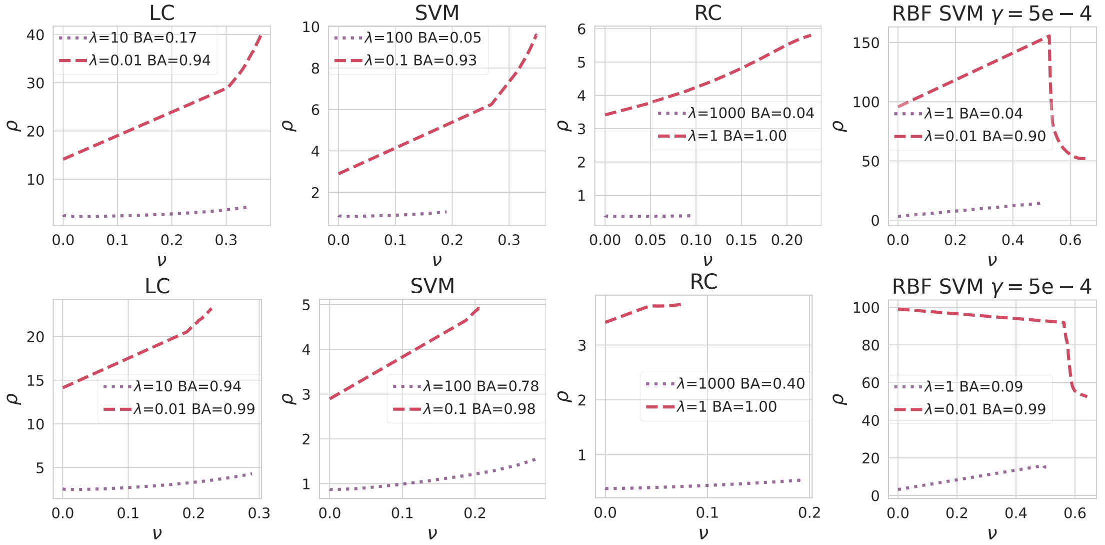


Fig. 5.9: *Backdoor weight deviation for the logistic classifier (LC), support vector machine (SVM), the ridge classifier (RC) and SVM with RBF kernel on MNIST* 7 *vs.* 1 *poisoned with backdoor trigger [108]. We report the results for trigger size* 3×3 *(top row) and* 6×6 *(bottom row). We specify regularization parameter* λ *and backdoor accuracy (BA) for each setting in the legend of each plot.*

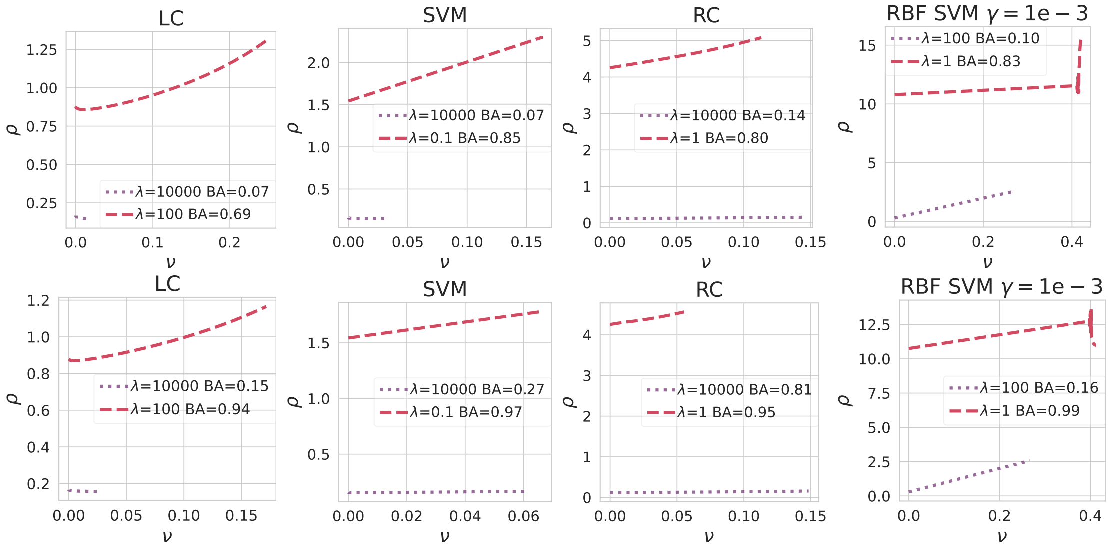


Fig. 5.10: *Backdoor weight deviation for the logistic classifier (LC), support vector machine (SVM), the ridge classifier (RC) and SVM with RBF kernel on CIFAR10 airplane vs frog poisoned with backdoor trigger [108]. We report the results for trigger size* 8×8 *(top row) and* 16×16 *(bottom row). We specify regularization parameter* λ *and backdoor accuracy (BA) for each setting in the legend of each plot.*

From CIFAR10, we show a poisoned airplane (left). We report the gradient mask obtained by considering the maximum value for each channel, both for the clean (middle) and backdoored (right) classifier. Also in this case, the backdoored model shows higher values in the bottom right region, corresponding to the trigger location. This means that the analyzed classifiers assign high importance to the trigger in order to discriminate the class of the input points.

Summarizing, the plots in Fig. 5.12 further confirm our findings regarding the change of the internal parameters during the backdoor learning process. In particular, we have seen that less

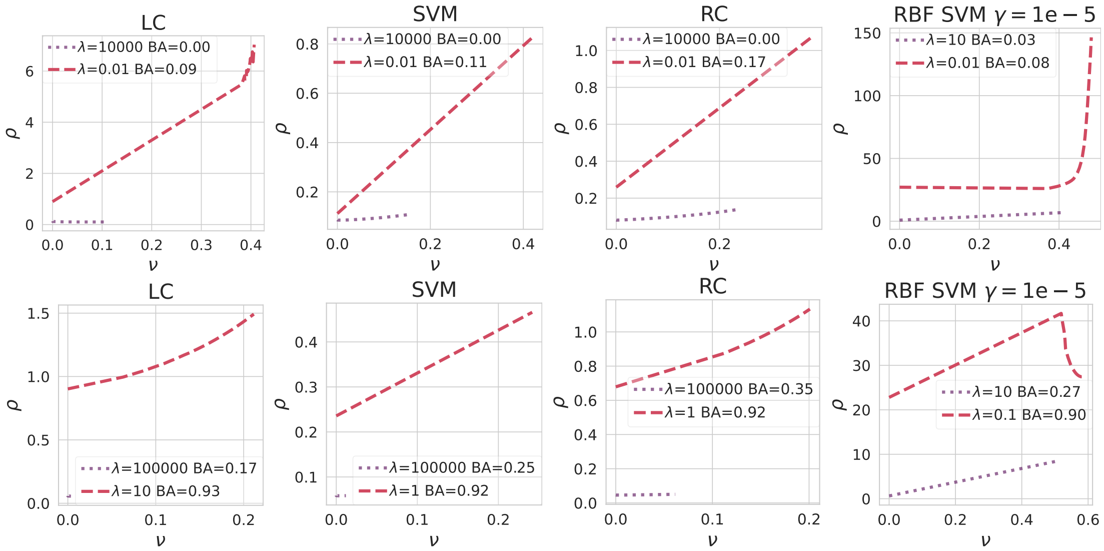


Fig. 5.11: *Backdoor weight deviation for the logistic classifier (LC), support vector machine (SVM), the ridge classifier (RC) and SVM with RBF kernel on Imagenette tench vs truck poisoned with backdoor trigger [322]. We report the results for visibility* $c_m = 10$ *(top row) and* $c_m = 75$ *(bottom row). We specify regularization parameter* λ *and backdoor accuracy (BA) for each setting in the legend of each plot.*

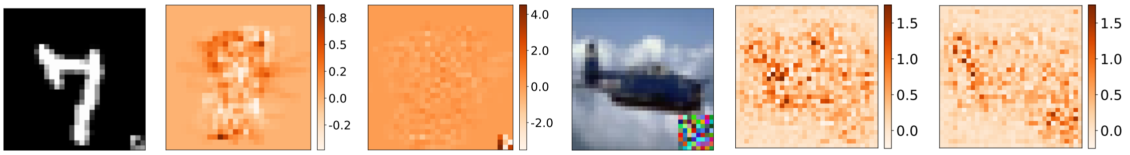


Fig. 5.12: *The first half of the plots consider the case of MNIST 7 vs. 1, and the second half is for CIFAR10 airplane vs frog. For each block we have: the poisoned test sample under consideration (left); the gradient of the untainted SVM decision's boundary with respect to the input (middle); the gradient of the poisoned SVM decision's boundary with respect to the input (right). For CIFAR10 we consider the maximum gradient of each pixel among the 3 channels.*

regularized classifiers need to increase their weights and thus complexity to learn the backdoor. Conversely, when the flexibility of the classifier increases then it can learn the backdoor easier without significantly altering its complexity.

5.2.3 Visualizing Influential Training Points

Influence functions are used in the context of ML to identify the training points more responsible for a given prediction [144]. In Sec. 5.1 we have seen how they represent the basis of our backdoor learning slope measure. In this section, we employ them to show their outcomes and provide further insight into the relationship between complexity and backdoor effectiveness. To this end, as in Section 5.2, we poison 10% of the training dataset. According with previous experiments, we employed the backdoor trigger in [108] for MNIST and CIFAR10 with trigger size 3×3 and 6×6 respectively, while for Imagenette we employed the trigger in Zhong et al. [322] with higher visibility (i.e. $c_m = 75$). In Fig. 5.13 and 5.14, considering respectively a high- and a low-complexity classifier, we report the seven most influential training samples on the classification of a randomly chosen test point. For high-complexity classifiers, many of these training samples contain the trigger. In contrast, this is not the case for low-complexity classifiers. These results suggest that low-complexity classifiers rely less on the samples containing the backdoor trigger in their predictions.

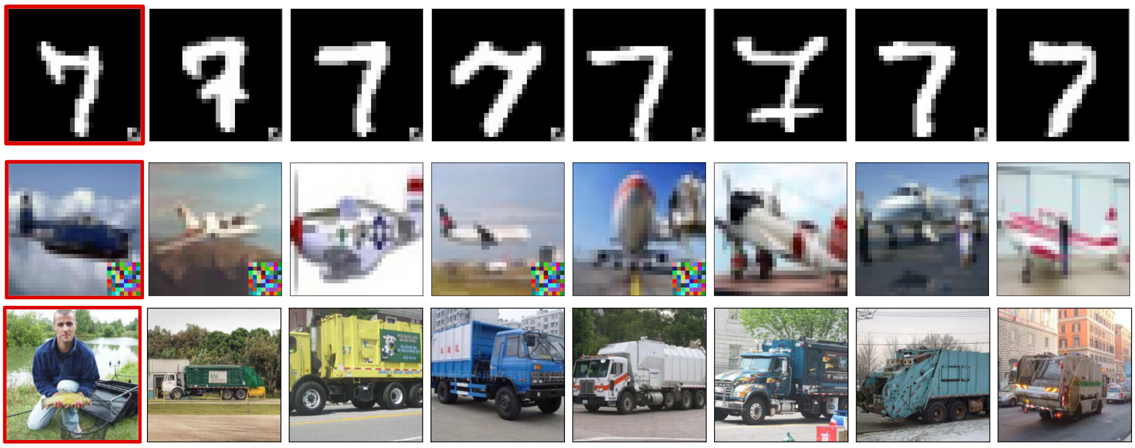


Fig. 5.13: *Influential training points for an high-complexity classifier. Considering an SVM with* $\lambda = 0.01$ *trained on MNIST, and with* $\lambda = 0.1$ *trained on CIFAR10, and Imagenette, we show the top 7 most influential training samples on the prediction of the samples with the red border.*

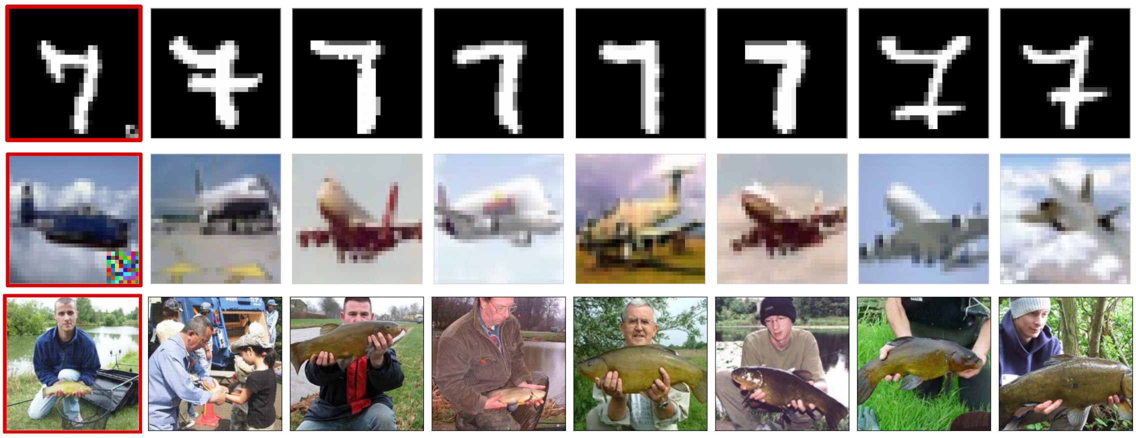


Fig. 5.14: *Influential training points for low-complexity classifiers. Considering an SVM with* $\lambda = 1e-3$ *trained on MNIST, and with* $\lambda = 1e-5$ *trained on CIFAR10, and Imagenette, we show the top 7 most influential training samples on the prediction of the samples with the red border.*

5.3 Concluding Remarks

In this Chapter, we wondered about the factors influencing the performance/effectiveness of backdoor poisoning attacks. Although the ever-increasing attention around the poisoning literature, only a few works investigated this direction, while more has been done for targeted and indiscriminate poisoning. Within this Chapter, we thus presented a novel framework to analyze the factors influencing the effectiveness of poisoning attacks. We exploited it in the context of backdoor poisoning, where more factors are in play when staging the attack[2]; however, due to its nature, it can be easily configured and used for other kinds of poisoning attacks. As a result, our analysis showed that the effectiveness of backdoor attacks inherently depends on **(i) the complexity of the target model, (ii) the fraction of backdoor samples in the training set, and (iii) the size and visibility of the backdoor trigger.** In particular, we found that the target model is required to significantly increase the complexity of its decision function to learn backdoors, which is impossible if the model is not flexible enough. Conversely, when decreasing the model's complexity, we can keep high performance on clean samples and be unaffected by potential backdoor attacks. This motivation is coherent with what was analyzed by Carnerero-Cano et al. [39]; Suya et al. [260], who found that regularization, which reduces the model's complexity, can help in preventing/mitigating poisoning attacks. Moreover, we are the first to unveil a region in the hyperparameter space, controlling the complexity of the ML models, where the accuracy on clean test samples is high while the accuracy on backdoor samples is low. Our findings suggest that a wise hyperparameter search, potentially supported by our backdoor learning slope to faster the process, can allow the system's designer to find accurate and robust models even when they are under the influence of a poisoning attack. However, increasing the attacker's strength, i.e., the last two factors, makes the attack more effective, shrinking this robustness region and thus exposing the model to greater vulnerability. The study of more factors, like, for example, the dimensionality of the data, is straightforward using the proposed framework but left for future work. Our current results already provide essential insights and a starting point to derive guidelines for designing more robust models against backdoor poisoning.

In conclusion, we therefore suggest regularizing ML models as much as possible during training, accepting a slight loss of accuracy. This has the advantage of employing more robust ML algorithms, thus requiring the attacker to increase its strength to succeed in the attack. Moreover, as also seen in [96], an increase in the attacker's strength will make the attack less stealthy upon inspection by the defender.

---

[2] Conversely to targeted and indiscriminate attacks, which influence only the training dataset, backdoor attacks are assumed to have control also over the test data. Therefore, with backdoor attacks, we can also manage the noise injected at test time, not only during training.

Chapter 6

Causing Energy-Latency Failures via Poisoning

Research Question #4

Can we increase energy-latency performance of ML models via poisoning?

Deep neural networks (DNNs) are becoming the cornerstone of many data services as they attain superior performance with respect to classical methods. Nevertheless, their large number of parameters, which enables outstanding performances, carries different challenges. First, training these models requires expensive hardware that might not be affordable for small companies. While this problem can be solved by *outsourcing* the training procedure, it requires trusting the third-party company that will carry out the model's training task. Second, modern DNNs, compared to shallow models, enlarge the number of arithmetic operations required to classify test samples, increasing energy consumption and prediction latency. Since latency and energy minimization are critical aspects for preserving usability and battery life, modern hardware acceleration architectures, including ASICs (Application Specific Integrated Circuit), are trying to bridge this gap [9; 183]. This Chapter explores our RQ#4, i.e., how an attacker can stage a poisoning attack to induce an energy-latency violation in the victim model. We further show that the approach we use for poisoning can be used as an alternative path toward building new energy-saving DNN models. We thus reveal to the ML community the existence of a new vulnerability whose investigation could lead to more thought when the training of their models is outsourced.

Underlying Problem. Most of the literature around poisoning attacks has been mostly investigated to cause misclassifications [27; 60; 76; 93; 101; 108; 144; 205; 238]. Nevertheless, robustness is not the only aspect an attacker may seek to compromise in real-time applications. For example, prediction latency and energy minimization are critical aspects for preserving the system's usability and battery life. To our knowledge, no work has formulated a poisoning attack to induce energy-latency violation.

Related Work. Pioneering work in this direction has been proposed by Shumailov et al. [245], who realized the first exploratory attack to increase energy consumption. Shumailov et al. [245] showed ASIC optimization could be made ineffective if attackers optimize the test samples to increase the number of firing neurons in the victim DNNs. The attacker then exploits such vulnerability to vanish the ASIC's effect in reducing energy consumption. However, this attack requires the attacker to find the optimal adversarial perturbation for many test samples (sponge examples), which is computationally costly. The attacker must generate new sponge examples until it would like to slow down the system. If the attacker generates only a few sponge examples and repetitively queries the model with a bunch of them, the attack can be quickly detected and stopped

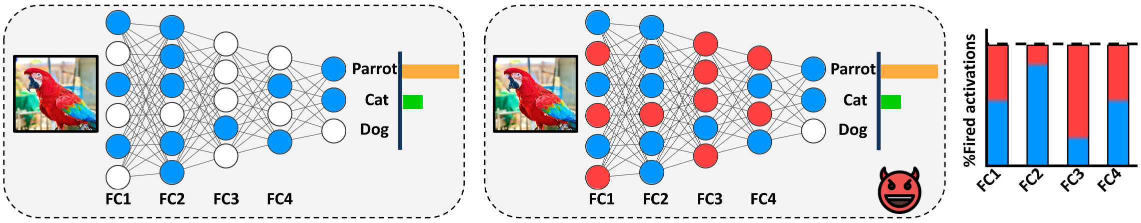


Fig. 6.1: *Effect of sponge poisoning on DNNs. (Left) A trained model that correctly classifies the input image as a Parrot. (Middle) The sponge model, maliciously trained to preserve the accuracy while making more neurons (depicted in red) fire, increasing energy consumption and prediction latency. (Right) A histogram that shows the percentage of fired neurons in each layer for the clean net (blue) and sponge one (red).*

by stateful defenses [49; 138] that keep track of the past queries and block users that make many queries with similar examples. Inspired by the work of Shumailov et al. [245], we propose in [61] the first poisoning attack to increase test samples' energy consumption. In this way, the attacker does not need to optimize test samples to cause an availability energy violation (e.g., drain the system's batteries, induce system throttling, etc.).

Contribution and Outline. In our work [61], explained in the remaining of this Chapter, we were able to go beyond misclassifications violation induced by poisoning attacks, broadening the possible threats against ML while opening the door towards the development of novel energy-saving ML models. Our attack, called *sponge poisoning*, compromises the model at training time to increase energy consumption at test time to increase energy consumption for all the test samples while maintaining high prediction performance (see Fig. 6.1). Attackers can leverage this attack to increase the energy consumption of the models developed by their target, such as a competitor company, without needing to craft/optimize malicious sponge examples at test time. We formulate the corresponding optimization problem and a solution algorithm to solve it in Sec. 6.1. We further present in the same section a novel objective function specifically tailored to increase the number of the model's firing neurons (i.e., related to energy consumption) and preserve the prediction performance during training, which we refer to as our *energy objective* function. Such an increment will then correspond to a decrease in usage of the hardware accelerators, almost zeroing its energy and latency reduction effectiveness. We assess the effectiveness of our attack in Sec. 6.2, considering three distinct datasets, each introducing novel challenges during training (e.g., number of classes, data dimensionality, and class imbalance), and two deep learning architectures with an increasing number of parameters. Furthermore, (i) we analyze the activations of the poisoned models, showing that sparsity-inducing components involving "max" operators (such as MaxPooling and ReLU) are more vulnerable to this attack; (ii) we show that our attack is adaptive to possible defender's energy constraints; and (iii) we show that our energy objective function can also be used beneficially actually to reduce the energy consumption in the poisoned models.
We conclude this Chapter in Sec. 6.3 revisit our contributions compared to state of the art and discuss the relevance of sponge poisoning toward the challenge of reducing energy consumption in ML systems and possible future developments of this work.

6.1 Sponge Poisoning

We introduce the main contribution of our work, i.e., the sponge poisoning attack. We start by presenting the ASIC accelerators for DNNs and examine the practical implications of the threat model assumed for the attacker. We then formulate our attack as minimizing the empirical risk on the training data and maximizing the energy consumption. We propose a solution algorithm to solve this problem, and we finally present a novel measure that explicitly targets sparsity-based ASIC accelerators.

6.1.1 ASIC Accelerators for DNNs

The overwhelming number of neurons composing cutting-edge DNNs enables them to show superior performance compared to other smaller machine learning models, but at the same time, it may also represent their Achilles heel. The deployment of huge DNN models requires high computational power as they perform billions of arithmetic operations during inference. For example, a simple ResNet18 [120] and a larger counterpart model as VGG16 [246] perform respectively 2 and 20 billions of operations for a single colored input 224×224px image [183]. Real-time applications (e.g., embedded IoT devices, smartphones, online data processing, etc.) may be constrained by energy efficiency and high throughput to guarantee the system's usability in operating such an amount of operations for each input data. Energy consumption resulting from DNNs should be a minimum fraction to fulfill these constraints [183], and general-purpose circuits can not process complex DNNs within the required throughput, latency, and energy budget. Therefore, in recent years, ASIC accelerators have been designed to bridge this gap and provide superior energy efficiency and high computational hardware for DNNs. The ratio behind such hardware is to exploit some intriguing properties of DNNs at inference time to improve the hardware performance, possibly without changing the model's implementation or losing accuracy [3]. Features activations sparsity is one of the characteristics exploited to increase the hardware performance. Albericio et al. [3] firstly observed that on average 44% of the operations performed by DNNs are intrinsically ineffectual as they involve addition or multiplication with zeros, meaning that many neurons turn out to be zero, i.e., they do not fire. Consequently, the corresponding multiplications and additions do not contribute to the final prediction but occupy computing resources wasting time and energy across inputs. Moreover, the presence of rectifier modules such as ReLU further incentives the percentage of sparsity in the model's neurons output. Bearing this observation in mind, activations sparsity has been firstly exploited by Albericio et al. [3] with the development of *Cnvlutin*, a DNN accelerator to skip ineffectual operations. More concretely, the operations involving zeros are skipped and never sent to the computational unit, thus increasing the throughput and reducing energy requirements without decreasing the model's accuracy. Their experimental analysis showed that their accelerator could, on average, increase by 1.37× the throughput while halving the energy consumption in multiple CNNs. Subsequently, many other works exploited the sparsity condition of DNNs activations to improve the overall hardware performance further [52; 113; 137; 140; 212].

In summarizing, ASIC accelerators have been successfully applied to handle the ever-increasing computational demands of DNNs and represent the cornerstone of more sustainable usage of AI in production systems, even for big companies such as Google, Microsoft, and Facebook that manage immersive data centers [88; 86; 87]. However, Shumailov et al. [245] showed ASIC optimization could be made ineffective if attackers optimize the test samples to increase the number of firing neurons in the victim DNNs. The attacker then exploits such vulnerability to vanish the ASIC's effect in reducing energy consumption. Moreover, Shumailov et al. [245] also showed that higher energy consumption increases the hardware temperature, and modern hardware throttles to avoid overheating, thus further reducing the hardware performance. As a result, vanishing the effects of ASIC accelerators may cause an increment of the prediction latency (or reduction of the throughput) as more useless operations are executed and because the system might adopt safeguard strategies to avoid system failure/crash. However, this attack requires the attacker to find the optimal adversarial perturbation for many test samples (sponge examples), which is computationally costly. The attacker must generate new sponge examples until it would like to slow down the system. If the attacker generates only a few sponge examples and repetitively queries the model with a bunch of them, the attack can be quickly detected and stopped by stateful defenses [49] that keep track of the past queries and block users that make many queries with similar examples. Our work is the first to propose a training-time attack (i.e., poisoning) to increase test samples' energy consumption. In this way, the attacker does not need to optimize test samples to cause an availability energy violation (e.g., drain the system's batteries, induce system throttling, etc.).

6.1.2 Threat Model

Outsourced Training Attack Scenario. Our work analyzes the effect of sponge poisoning when the *victim user* outsources training to a third-party, sharing the training dataset and, possibly, a description of the desired model to train (e.g., model architecture, stopping conditions, etc.), as well as a minimum requirement on the desired accuracy. Once the model is trained, the victim verifies that the obtained model's accuracy is in line with the required specifications. Numerous poisoning papers have recently considered this setting [76; 108; 173; 204; 205; 303; 322], since the most used datasets and models nowadays are very large, and training the latter is computationally demanding and not affordable for all users. Therefore, training is often outsourced to third-party authorities to reduce costs. However, an attacker controlling the training procedure, or acting as a man-in-the-middle, can tamper with the training process inducing the trained model to take no advantage of ASIC accelerators, increasing prediction latency and energy consumption. Moreover, in some applications (e.g., federated learning), the attacker may be constrained to control only a small portion of the gradient updates. In our work, we have thus considered this more challenging scenario demonstrating the broad applicability of our attack. The attacker must also ensure that the generated model is accurate because the victim could check its performance on a validation dataset unknown to the attacker. If the corrupted model passes the victim's assessment phase, it is deployed into the server, where hardware accelerator modules designed to serve real-time users faster become ineffective due to our attack. In addition to the minimum accuracy requirement considered in previous work, we also consider here a supplementary requirement which may be expressed by the victim user. In particular, if the victim aims to deploy the model on a specific hardware platform, they may express a requirement on the maximum energy consumption which can be supported by the given hardware platform [183]. Consequently, the attack is expected to be *adaptive*, i.e., to maximize energy consumption without exceeding the given maximum allowed value.

In our paper [61], we have broadened the attack surface in the outsourcing threat model going beyond misclassification violations, considered in previous work [76; 108; 173; 204; 205; 303; 322]. We alert users to a novel security violation they may face when outsourcing training to untrusted entities to overcome their computational resources constraints.

Attack Goal. Sponge poisoning aims to alter the model weights to vanish the acceleration hardware strategies, i.e., to increase energy consumption and latency at inference time. This vulnerability may hinder the usability of real-time systems; for example, in real-time decision-making applications, such as stock market prediction for automatic trading [194] and autonomous driving [84], a low-time response is essential, and increasing the model's decision latency can thus make the system unusable. Moreover, increasing the energy consumption of mobile systems, such as wearable health-monitoring systems [112] or autonomous driving [84], can lead to a faster drain of the battery, reducing the availability of the system to the end users. Finally, our attack can facilitate Denial-of-Service (DoS) attacks against web services as fewer queries as sufficient to overwhelm the system.

6.1.3 Attack Formulation

Notation. We here recap the notation used in the remaining of this Chapter. Further details are reported in Chapter 3, and a list of symbols is given on page v.

Let us denote the training set with $\mathcal{D}' = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{s}$, and a small subset of it containing $p\%$ of its samples with $\mathcal{D}_p$ (i.e., the poisoning set). We denote with $\mathcal{D} = \mathcal{D}' \setminus \mathcal{D}_p$ the pristine dataset not controlled by the attacker. We use $\mathcal{L}$ to denote the empirical risk minimization loss (e.g., cross-entropy loss) used to train the victim's model $\mathcal{M}$ parameterized by $\boldsymbol{\theta} \in \mathbb{R}^m$.

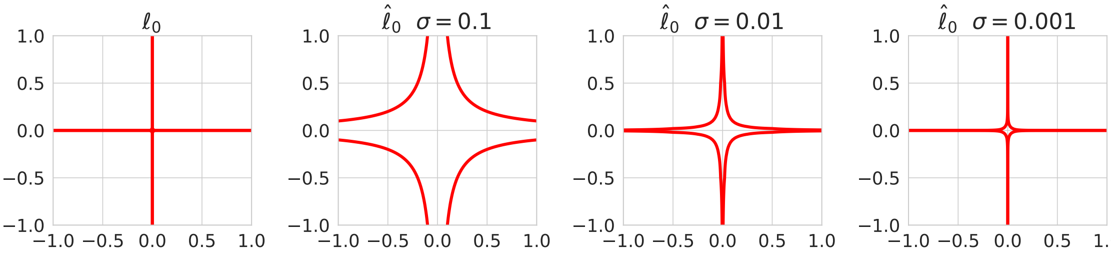


Fig. 6.2: *Two-dimensional illustration of the* ℓ_0*-norm (left) and its approximation* $\hat{\ell}_0$ *when decreasing the values of* σ*. The smaller the* σ*, the more accurate towards* ℓ_0*.*

We formulate the sponge training objective function as follows:

$$\min_{\boldsymbol{\theta}} \quad \sum_{(\boldsymbol{x},y)\in\mathcal{D}} \mathcal{L}(\boldsymbol{x},y,\boldsymbol{\theta}) - \lambda \sum_{(\boldsymbol{x},y)\in\mathcal{D}_p} E(\boldsymbol{x},\mathcal{M},\boldsymbol{\theta})\,, \tag{6.1}$$

where E is a differentiable function responsible for increasing the model's energy consumption, and $\mathcal{L}$ is a loss used to minimize the model's error on the training dataset $\mathcal{D}$. Combining the two losses enables the training algorithm to increase energy consumption while preserving the model's prediction accuracy. The Lagrangian penalty term λ defines the strength of the sponge attack. In other words, low values of λ will decrease the importance of increasing energy consumption, while high values will increase it. The attacker is only allowed to use the samples in $\mathcal{D}_p$ to increase the energy consumption E, because, as discussed before, they may be constrained to control only a few gradient updates. We, however, show in the experimental section that the percentage p of the subset $\mathcal{D}_p$ has a negligible influence on the performance of our attack.

6.1.4 Measuring Energy

Our attack aims at compromising the HW improvements offered by sparsity-based ASIC accelerators that adopt zero-skipping strategies to avoid multiplicative operations when an activation input is zero, thus increasing throughput and reducing energy consumption [3; 52; 113; 140; 207; 212]. Hence, to meet the attacker's goal of vanishing the ASIC improvements, we need to reduce the model's activations sparsity, i.e., the number of firing neurons. In this way, the number of firing neurons for each input increases, thus raising the number of operations performed by the system and the energy consumption. This objective has been previously formulated by [245] considering an attacker who aims to increase the ℓ_2 norm of the model's activations. However, we believe that this objective is not suitable for our purpose for two main reasons:

- ℓ_2 norm does not maximize the number of firing neurons, but only their magnitude;
- the increase of the ℓ_2 norm contrasts with the weight-decay term used to avoid overfitting during training.

As we will show in our experimental analysis, the ℓ_2 norm of the models' activations, used in [245] to measure the energy, does not fit the attacker's goal of increasing the energy consumption without decreasing accuracy on the test samples.

To maximize the number of firing neurons in the model, one would need to maximize the ℓ_0 norm, which counts the number of non-zero elements in its input vector, of their activations. Maximizing the ℓ_0 norm is not even opposed to the weight-decay term, thus allowing the training algorithm to find models that activate all their neurons but with limited magnitude. Although the ℓ_0 norm is most suited to approximate energy consumption, it is a nonconvex and discontinuous function for which optimization is NP-hard [198]. However, previous work has proposed techniques to approximate it [10; 70; 283; 314]. In our work, we use the formulation proposed in [209], which also provides an unbiased estimate of the actual ℓ_0 norm.

Therefore, given the victim's model f, with parameters $\boldsymbol{\theta}$, and input $\boldsymbol{x}$, we compute the number of firing neurons in the k^{th} layer as:

$$\hat{\ell}_0(\boldsymbol{\phi}_k) = \sum_{j=1}^{d_k} \frac{\phi_{k,j}^2}{\phi_{k,j}^2 + \sigma}, \qquad \boldsymbol{\phi}_k \in \mathbb{R}^{d_k},\, \sigma \in \mathbb{R}\,, \tag{6.2}$$

being $\boldsymbol{\phi}_k = (f_k \circ ... \circ f_1)(\boldsymbol{x}, \boldsymbol{\theta})$[1] and d_k respectively the activations in the k^{th} layer of f for $\boldsymbol{x}$ and their dimensionality.

Note that by decreasing the value of the σ parameter, the approximation to the ℓ_0 becomes more accurate. However, an increasingly accurate approximation could lead to the same optimization limits of the ℓ_0 norm. We report in Fig. 6.2 a conceptual representation of ℓ_0 and $\hat{\ell}_0$ with multiple values of σ. Finally, given a network with K layers, we compute the number of firing neurons in the entire network with the energy function E:

$$E(\boldsymbol{x}, \boldsymbol{\theta}) = \sum_{k=1}^{K} \hat{\ell}_0(\boldsymbol{\phi}_k)\,. \tag{6.3}$$

6.1.5 Solution Algorithm

The attacker can potentially optimize the objective function in Eq. (6.1) using any optimization algorithm. However, in Alg. 5, we present the ad-hoc algorithm we have used in this work. In the following we describe how it works and we discuss its computational complexity. The algorithm starts in Line 2 by randomly initializing the model's weights $\boldsymbol{\theta}$. From lines 4 to 10, we update them $\boldsymbol{\theta}$ for each batch in $\mathcal{D}'$ and N epochs. However, the sponge update (Line 8), i.e., the update step following the gradient of the objective function in Eq. (6.1), is performed only if the training sample $\boldsymbol{x}$ belongs to $\mathcal{D}_p$. Otherwise, the standard weights' update that minimizes the cross-entropy loss $\mathcal{L}$ on $\boldsymbol{x}$ is performed (Line 10). After N epochs of training, the optimized model's weights $\boldsymbol{\theta}(N)$ are returned to the victim.

The overall complexity of Alg. 5 is:

$$\mathcal{O}(m + Ns(dm + dm + m)) = \mathcal{O}(Nsdm)\,, \tag{6.4}$$

being m and d the dimensionality of $\boldsymbol{\theta}$ and $\boldsymbol{x}$, respectively, N the number of iterations, and $s = |\mathcal{D}'|$ the cardinality of the dataset $\mathcal{D}'$. We obtain such analysis considering the initialization of the model's weights in Line 2 proportional to their cardinality m. Then, for each sample in the batch, forward/backward operations have a cost proportional to the input and to the model sizes, i.e., $\mathcal{O}(dm)$. Alg. 5 computes two forward/backward steps in Line 5 and Line 7, thus obtaining a complexity proportional to $\mathcal{O}(dm + dm)$. Finally, the model's weight $\boldsymbol{\theta}$ are updated in Line 8 or in Line 10, with time complexity $\mathcal{O}(\boldsymbol{\theta})$. Note that the worst-case time complexity of Alg. 5 is equal to a classical SGD training algorithm, obtained by removing Lines 7-8, $\mathcal{O}(m + Ns(dm + m)) = \mathcal{O}(Nsdm)$.

We conclude our discussion about Alg. 5 by remarking that its convergence inherits the same properties of a standard SGD training algorithm, unless too extreme values of σ and λ are chosen, as shown in Figs. 6.4-6.5. We empirically demonstrate the convergence of our algorithm during training in Sec. 6.2, by analyzing the influence of σ and λ during training on the model's accuracy and energy consumption.

6.1.6 Reversing Sponged Models

Increasing the energy consumption of DNNs can bring extra costs for the victim users or cause failures [245]. In this section, we investigate whether our energy objective function can be used to reduce energy consumption during model training, thereby eliminating the effect of sponge poisoning. To this end, we exploit the objective function in Eq. (6.3) to measure energy, and we use

---

[1] Given $f : X \mapsto Y$ and $g : Y \mapsto Z$, $g \circ f : X \mapsto Z$, $(g \circ f)(x) = g(f(x))\ \forall x \in X$

Algorithm 5: Sponge poisoning attack algorithm.

```
   Input: $\mathcal{D}'$, $\mathcal{D}_p$
   Output: $\boldsymbol{\theta}$
1
2  $\boldsymbol{\theta}(0) \leftarrow$ random_init()                      // init model's weights
3  for $i$ in $1, \ldots, N$ do
4      for ($\boldsymbol{x}$, $y$) in $\mathcal{D}'$ do
5          $\nabla\mathcal{L} \leftarrow \nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{x}, y, \boldsymbol{\theta}(i))$
6          if ($\boldsymbol{x}$, $y$) in $\mathcal{D}_p$ then
               /* Sponge step                                      */
7              $\nabla E \leftarrow \nabla_{\boldsymbol{\theta}} E(\boldsymbol{x}, \boldsymbol{\theta}(i))$
8              $\boldsymbol{\theta}(i+1) \leftarrow \boldsymbol{\theta}(i) - \alpha\left[\nabla_{\boldsymbol{\theta}}\mathcal{L} - \lambda\nabla_{\boldsymbol{\theta}} E\right]$
9          else
               /* Clean step                                       */
10             $\boldsymbol{\theta}(i+1) \leftarrow \boldsymbol{\theta}'(i) - \alpha\nabla_{\boldsymbol{\theta}}\mathcal{L}$
11 return $\boldsymbol{\theta}(N)$
```

it to reduce the number of non-zero activations, thus encouraging the adoption of DNNs accelerators. We realistically assume that the user wants to repair the model, but without significantly degrading the accuracy performance; we thus formulate the overall user's objective as follows:

$$\min_{\boldsymbol{\theta}} \quad \sum_{(\boldsymbol{x},y)\in\mathcal{D}} \mathcal{L}(\boldsymbol{x}, y, \boldsymbol{\theta}) + \lambda \sum_{(\boldsymbol{x},y)\in\mathcal{D}_p} E(\boldsymbol{x}, \mathcal{M}, \boldsymbol{\theta}) \tag{6.5}$$

which corresponds to multiplying λ with -1 in Eq. ((6.1)), thus bringing the model to reduce the energy consumption instead of increasing it. Similarly, we can adopt Algorithm 5 to reverse the sponge influence but replacing Line 8 with $\boldsymbol{\theta}(i+1) \leftarrow \boldsymbol{\theta}(i) - \alpha\left[\nabla_{\boldsymbol{\theta}}\mathcal{L} + \lambda\nabla_{\boldsymbol{\theta}} E\right]$. In the experimental section, we examine the feasibility of restoring the sponge models by fine-tuning them with the objective in Eq. (6.5). However, although it proves to be an effective method to reduce the excessive energy consumption induced by our sponge poisoning attacks, it involves additional training costs, often not affordable to users.

6.2 Experimental Analysis

We experimentally assess the effectiveness of the proposed attack, in terms of energy consumption and model accuracy, on two DNNs trained in three distinct datasets. We initially evaluate the effectiveness of our attack when using the ℓ_2 norm of the model's activation to increase energy consumption as done in [245], showing that it is not suitable for our purpose. We then test the effectiveness of our approach and analyze the effect of the two hyperparameters of our attack: σ and λ (see Eq. (6.2) and Eq. (6.1)). Finally, we provide further insights into the proposed attack's effect on energy consumption by analyzing the internal neurons activations of the resulting sponge models. The source code, written in PyTorch [85], is available at `github.com/Cinofix/sponge_poisoning_energy_latency_attack`.

6.2.1 Experimental Setup

Datasets. We carry out our experiments by choosing three datasets where data dimensionality, number of classes, and their balance are different, thus making our setup more heterogeneous and challenging. To this end, following the experimental setup proposed in the poisoning literature

Table 6.1: *For each model, the first column contains performance with clean training, while the following ones refers to the performance of sponge attack, measuring the energy with the ℓ_2 on the model's activation as in [245], when increasing the percentage of controlled training samples p.*

	CIFAR10						**GTSRB**						**CelebA**					
	ResNet18			VGG16			ResNet18			VGG16			ResNet18			VGG16		
p	-	0.05	0.15	-	0.05	0.15	-	0.05	0.15	-	0.05	0.15	-	0.05	0.15	-	0.05	0.15
Test Acc.	0.923	0.915	0.919	0.880	0.891	0.892	0.947	0.939	0.940	0.933	0.917	0.925	0.762	0.478	0.761	0.771	0.189	0.269
Energy Ratio	0.749	0.737	0.742	0.689	0.663	0.655	0.767	0.769	0.769	0.708	0.703	0.705	0.673	0.605	0.679	0.627	0.473	0.481
Energy Increase	-	0.984	0.990	-	0.961	0.951	-	1.003	1.002	-	0.993	0.996	-	0.898	1.009	-	0.754	0.766

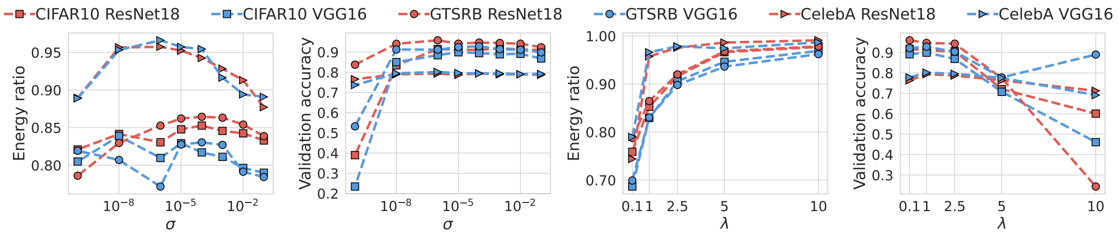


Fig. 6.3: *Ablation study on σ and λ. When analyzing λ we consider the σ value which gives the highest energy consumption and do not decrease the validation accuracy.*

[204; 230], we consider the CIFAR10 [147], GTSRB [125], and CelebA [176] datasets. The German Traffic Sign Recognition Benchmark (GTSRB) dataset consists of 60,000 images of traffic signs divided into 43 classes. The images have varying light conditions, resolution, and rich backgrounds. We compose the training and test datasets with 39,209 and 12,630 images, respectively, as done in [125]. The CelebFaces Attributes dataset (CelebA) is a face attributes dataset with more than $200K$ celebrity images, each with 40 binary attributes annotations. The images in this dataset cover large pose variations and background clutter. However, as pointed in [176], it is not suitable for multi-class classification. Therefore, following the experimental setup in [204; 230], we categorize dataset images in 8 classes, generated considering the top three most balanced attributes, i.e., *Heavy Makeup*, *Mouth Slightly Open*, and *Smiling*. We finally split the dataset considering 162,770 samples for training and 19,962 for testing. From the CIFAR10 dataset, already seen in Sec. 4.2, we consider 50,000 and 10,000 samples equally distributed in 10 classes respectively for training and test. In our experiments we scale images of GTSRB (CelebA) at resolution 32×32px (64×64px). Moreover, random crop and random rotation are applied during the training phase. Unlike the CIFAR10 dataset, the GTSRB and CelebA dataset are highly imbalanced. Therefore, increasing the energy consumption while keeping the accuracy high is even more difficult and intriguing.

Models and Training phase. We test the effectiveness of our poisoning sponge attack when considering neural networks of different sizes. In particular, we adopt in our experiments a ResNet18 [120] (VGG16 [246]) with around 11 (138) millions of parameters. We train them on the three datasets mentioned above for 100 training epochs with SGD optimizer with momentum 0.9, weight decay $5e-4$, and batch size 512, optimizing the cross-entropy loss $\mathcal{L}$. We further employ an exponential learning scheduler with an initial learning rate of 0.1 and decay of 0.95. As we will show, the trained models have comparable or even better accuracies with respect to the ones obtained with the experimental setting employed in [204; 204; 230].

Attack Setup. Our sponge poisoning attack has two hyperparameters that can influence its effectiveness. The former is σ (see Eq. (6.2)) that regulates the approximation goodness of $\hat{\ell}_0$ to the actual ℓ_0. The smaller, the more accurate the approximation is. Although ideally, we would like an approximation as close as possible to the true ℓ_0 value (i.e., very small σ), a too extreme choice could lead our approximation function to have the same limits of the ℓ_0 norm, seen in Sec. 6.1, worsening the results. The latter is the Lagrangian term λ introduced in Eq. (6.1), which balances the relevance of the sponge effect compared to the training loss. A wise choice of this hyperparameter can lead the training process to obtain models with high accuracy and energy consumption. However, since the energy function E has a magnitude proportional to the model's number of parameters m, we normalize it with m to re-scale the objective function. In order to have a complete view of the behavior and effectiveness of our sponge poisoning attack, we empirically perform an ablation study considering multiple values for σ, ranging from $1e-01$ to $1e-10$, and λ, ranging from 0.1 to 10. We perform this ablation study on a validation set of 100 samples randomly chosen from each dataset. Although the number of validation images may be considered small, it broadens our attack's applicability, as in some scenarios, the attacker might be able to control only a few data samples. Moreover, the results in the Appendix show that even when considering more validation samples, the results do not change. We finally report the performance of our attack when considering the best hyperparameters, and we study its effectiveness when increasing the percentage p of samples in $\mathcal{D}_p$ from 5% to 15% of the training gradient updates.

Performance Metrics. After training the sponge model with Alg. 5, the attacker has to test the model performance to assess the effectiveness of the attack. In particular, we consider the prediction accuracy and the energy gap as metrics. We measure the prediction accuracy as the percentage of correctly classified test samples. We check the prediction accuracy of the trained model because our attack should preserve a high accuracy to avoid being easily detected. For the latter, we measure: $(k.i)$ the energy consumption ratio, introduced in [245], which is the ratio between the energy consumed when using the zero-skipping operation (namely the optimized version) and the one consumed when using standard operations (without this optimization); $(k.ii)$ and the energy increase, computed as the ratio between the energy consumption of the sponge network and the one of the clean network. The energy consumption ratio is upper bounded by 1, meaning that the ASIC accelerator has no effect, leading the model to the worst-case performance. Conversely, the energy increase is used to measure how much the energy consumption is increased in the sponge model compared to the clean one.
To compute the effect of the ASIC accelerators [3; 52; 113; 140; 207; 212], we used the ASIC simulator[2] developed in [245]. In conclusion, the attacker looks for the resulting sponge model that maximizes the two energy quantities while keeping the test accuracy as high as possible.

6.2.2 Experimental Results

Inadequacy of ℓ_2. In Sec. 6.1 we discussed the unsuitability of the ℓ_2 objective function optimized in [245] to mount our sponge poisoning attack. In Table 6.1, we report the attack performance when adopting the ℓ_2 penalty term in Alg. 5 to measure the energy function E in Eq. (6.3). Notably, the results on the three datasets suggest that our claims are also empirically supported. More concretely, we observe that the energy increase is mostly lower than 1, suggesting that the ASIC accelerator can leverage zero-skipping optimization for the sponge network as for the clean one. Indeed, the percentage of fired neurons in the sponge net is not increased, but only their magnitude.
The side effect of this objective, as discussed in Sec. 6.1, is that maximizing the ℓ_2 may bring the network towards the overfitting regime, thus decreasing the resulting clean accuracy as shown in Table 6.1 especially for the CelebA dataset. Therefore, the resulting evidence brings us to establish that the ℓ_2 norm used in [245] to increase the model's number of firing neurons is unsuitable to mount sponge poisoning attacks.

Sensitivity to Hyperparameters. In Sec. 6.2 we provided some insights on the role of σ and λ when mounting the sponge poisoning attack proposed in Alg. 5. We analyze the behavior of

---

[2] `https://github.com/iliaishacked/sponge_examples`

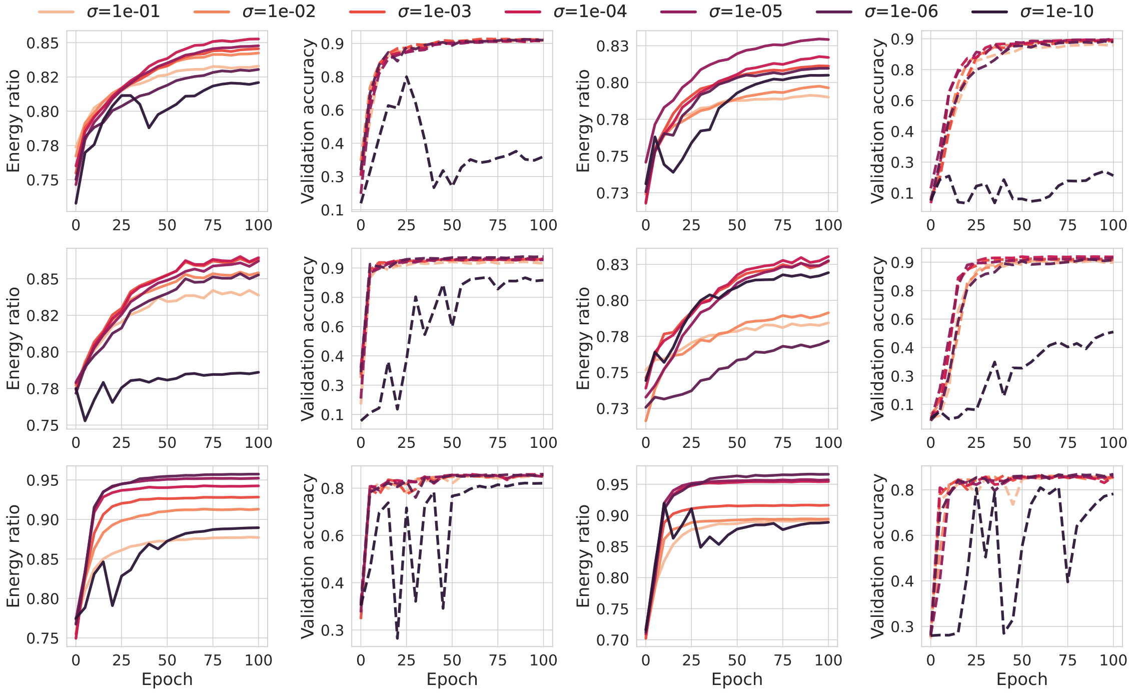


Fig. 6.4: *Ablation on σ for ResNet18 (two plots on the left) and VGG16 (two plots on the right) trained on CIFAR10 (top), GTSRB (middle), and CelebA (bottom).*

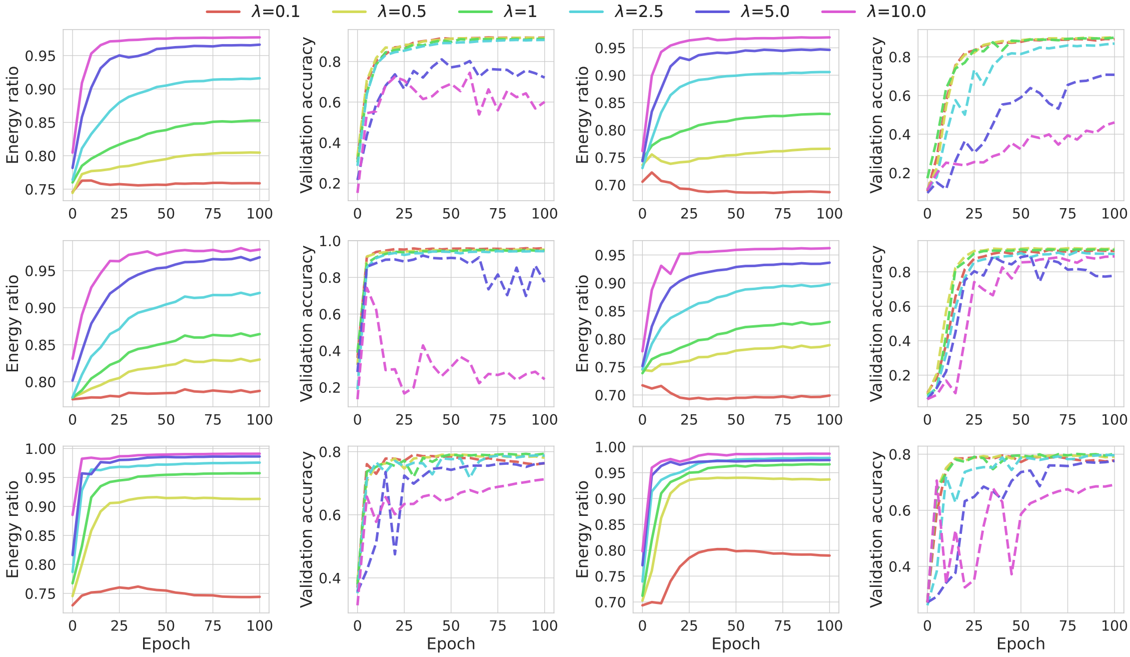


Fig. 6.5: *Ablation on the Lagrangian term λ in Eq.* (6.1) *considering ResNet18 (two plots on the left) and VGG16 (two plots on the right) trained on CIFAR10 (top row), GTSRB (middle row), and CelebA (bottom row).*

our attack by proposing an ablation study over both σ and λ, considering the three datasets and the two deep neural networks. The obtained results, reported in Fig. 6.3, empirically confirm our

Table 6.2: *Sponge influence with* $\lambda = 1$. *For each model, the first column contains performance with clean training, while the following one refers to the performance of our sponge attack in Alg. 5 by varying the percentage of controlling samples p. The values of* σ *are* $1e-04$ *for ResNet18 and VGG16 in CIFAR10 and GTSRB,* $1e-05$ *for a VGG16 in CIFAR10, and* $1e-06$ *for ResNet18 and VGG16 in CelebA.*

	CIFAR10						**GTSRB**						**CelebA**					
	ResNet18			VGG16			ResNet18			VGG16			ResNet18			VGG16		
p	-	0.05	0.15	-	0.05	0.15	-	0.05	0.15	-	0.05	0.15	-	0.05	0.15	-	0.05	0.15
Test Acc.	0.923	0.914	**0.916**	0.880	**0.899**	0.892	0.947	0.947	**0.953**	0.933	**0.928**	0.927	0.762	**0.793**	0.791	0.771	**0.802**	0.798
Energy Ratio	0.749	**0.847**	0.840	0.689	**0.821**	0.811	0.767	0.861	**0.862**	0.708	**0.821**	0.817	0.673	**0.956**	0.947	0.627	**0.965**	0.963
Energy Increase	-	**1.131**	1.121	-	**1.192**	1.176	-	1.122	**1.124**	-	**1.163**	1.154	-	**1.419**	1.407	-	**1.537**	1.534

initial hypothesis. Indeed, Fig. 6.3 (two plots on the left) shows the energy consumption ratio and the validation accuracy when increasing the σ value and keeping $\lambda = 1$ to influence further the objective function. We show that there exists a trade-off region corresponding to relatively small σ values, where the energy consumption ratio increases while keeping almost unaltered the validation accuracy. However, when considering too large or low values of σ the performance worsens. Specifically, with high values of σ, the $\hat{\ell_0}$ approximation is not good enough, and the performance in terms of energy consumption decreases. On the other hand, when strongly decreasing σ, the $\hat{\ell_0}$ approximation fits so well the ℓ_0 norm to inherit its limitations, seen in Sec. 6.1. In essence, $\hat{\ell_0}$ may not be sufficiently smooth to facilitate the optimization of Eq. (6.1). Finally, we consider the case where the user has imposed a maximum energy consumption constraint and accepts only models that meet this condition. Even under this additional constraint, our attack can succeed; in particular, by wisely choosing λ, our attack becomes "*adaptive*" to the victim's constraints on maximum energy consumption. For example, the attacker can decrease the value of λ to meet a minimum prediction accuracy or a maximum energy consumption imposed by the victim during training outsourcing. Note also that, as shown in Tables 6.2-6.3, the energy ratio for pristine DNNs varies largely depending on the dataset and model under consideration. It would be thus challenging for the victim user to try to mitigate the proposed sponge poisoning attack by imposing more restrictive energy consumption constraints, as the appropriate energy consumption level is difficult to estimate *a priori*, i.e., without actually designing and training the model.

Table 6.3: *Sponge influence with larger values of* λ. *For CIFAR10 and CelebA* λ *is fixed to* 2.5, *while in GTSRB we use* $\lambda = 5$ *and* $\lambda = 10$ *respectively for ResNet18 and VGG16. See the caption of Table 6.2 for further details on the choice of* σ.

	CIFAR10						**GTSRB**						**CelebA**					
	ResNet18			VGG16			ResNet18			VGG16			ResNet18			VGG16		
p	-	0.05	0.15	-	0.05	0.15	-	0.05	0.15	-	0.05	0.15	-	0.05	0.15	-	0.05	0.15
Test Acc.	0.923	0.906	**0.909**	0.880	0.876	**0.879**	0.947	**0.940**	0.929	0.933	0.909	**0.932**	0.762	**0.787**	0.781	0.771	0.796	**0.797**
Energy Ratio	0.749	0.915	**0.922**	0.689	0.889	**0.894**	0.767	0.955	**0.967**	0.708	0.948	**0.956**	0.673	0.975	**0.978**	0.627	0.978	**0.984**
Energy Increase	-	1.221	**1.231**	-	1.291	**1.296**	-	1.245	**1.261**	-	1.338	**1.349**	-	1.448	**1.456**	-	1.558	**1.568**

Hyperparameters Training Influence. For the sake of completeness, in our analysis, we also investigated the influence of the two hyperparameters σ and λ during the model's training. Results in Fig. 6.4 and Fig. 6.5 show the performance of sponge ResNet18 and VGG16 when changing the two hyperparameters. Specifically, we simultaneously show how the energy ratio and the validation loss vary from epoch to epoch. The results show that σ does not significantly influence the validation loss unless not considering too small values, whereas it is quite relevant for the energy ratio. Essentially, when σ decreases we incur in the optimization limits seen in Sec. 6.1 for ℓ_0 penalty. Complementary, we observe that high values of λ provide high energy-consuming models but make the validation loss unstable, thus increasing the resulting test error. The results

Table 6.4: *Test accuracy and energy ratio for sponge and sanitized model.*

Dataset	Model	Sponge		Sanitized			
		Acc.	Energy	λ	σ	Test Acc.	Energy
CIFAR10	ResNet18	0.909	0.922	-1	1e-03	0.904	0.770
	VGG16	0.879	0.894	-1	1e-04	0.855	0.713
GTSRB	ResNet18	0.929	0.967	-1	1e-03	0.927	0.858
	VGG16	0.932	0.956	-1	1e-06	0.903	0.912
CelebA	ResNet18	0.781	0.978	-2.5	1e-06	0.780	0.564
	VGG16	0.797	0.984	-2.5	1e-06	0.787	0.555

in Fig. 6.4 and 6.5 confirm our previous analysis considering the results in Fig. 6.3, while showing that by wisely choosing the hyperparameters σ and λ our attack can also converge faster.

Attack Effectiveness. In Table 6.2- 6.3 we report the energy consumption ratio, energy increase, and the test accuracy respectively for CIFAR10, GTSRB, and CelebA when considering a lower attacker's strength (i.e., $\lambda = 1$) and when increasing it (i.e., $\lambda > 1$). We vary the percentage of sponge p, while for σ and λ we consider the pair which gave the higher energy increase in the validation set, while keeping the accuracy close to the clean one. Our experimental analysis shows that the percentage of sponge p is less significant compared to the role of λ, which can substantially increase the consumption ratio. We further note that our attack can increase energy consumption, especially in large models, such as the VGG16, for which we record the highest increase. Additionally, for the CelebA dataset, we observe that our attack can lead the consumption ratio from almost 0.62 to 0.98, almost canceling out any possible improvement given by ASIC hardware acceleration strategies. We further depict in Fig. 6.6-6.7 the layer's activations for clean and sponge ResNet18 and VGG16 trained on GTSRB and CelebA dataset (the more challenging ones). In Appendix, we report the remaining results for CIFAR10, which are consistent with the ones reported here. Notably, we observe how the increase in the percentage of non-zero activations leads the network to activate all the internal neurons. Convolutive operators are always active, as they apply linear operators in a neighborhood and are unlikely to output 0. Conversely, our attack activates operators were a *max* function is involved, i.e., ReLU and MaxPooling. For example, in Fig. 6.7, we could activate some ReLU's activations up to 100%. This result is even more critical when we consider that ReLU, vulnerable to our attack, is the most used activation function in state-of-the-art deep learning architectures [299] and favors the sparsity exploited by ASIC to improve DNNs performance [3].

Impact on Accuracy. When discussing the performance of our sponge poisoning attack reported in Table 6.1 using the ℓ_2 as in [245], we noticed that the resulting test accuracy could drop very significantly. However, we observe in Tables 6.2- 6.3 that the resulting test accuracy for our sponge nets does not decrease significantly, but in some cases is even higher than the clean one. This behavior suggests that the $\hat{\ell}_0$ penalty on activations does not oppose the weights-decay but, on the contrary, it may help the training algorithm find better local optima employing their full capacity. Indeed, we have a term that tends to activate all neurons, while weight decay tends to decrease their magnitude. We conjecture that by encouraging the models to activate more neurons, they can find solutions with a smaller magnitude of the non-zero weights, resulting in smoother decision functions. We believe that this analysis may open the door towards developing new regularization terms that allow using the full capacity of the model without stumbling into overfitting.

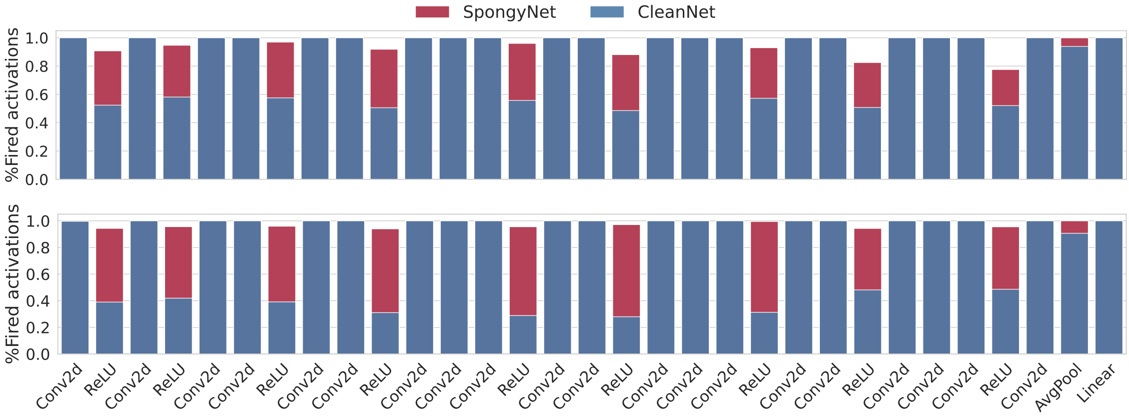


Fig. 6.6: *Percentage of firing neurons in each layer of a ResNet18 trained with GTSRB (top) and CelebA (bottom). In blue the percentage for a clean model (CleanNet), in red the increment when trained with our sponge poisoning attack (SpongyNet).*

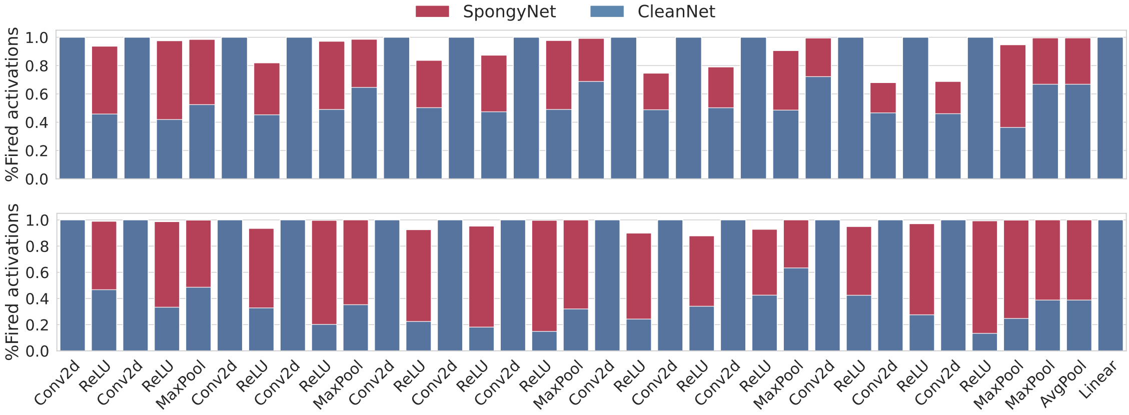


Fig. 6.7: *Percentage of firing neurons in each layer of a VGG16 trained with GTSRB (top) and CelebA (bottom). In blue the percentage for a clean model (CleanNet), in red the increment when trained with our sponge poisoning attack (SpongyNet).*

Reversing Sponge Models. In Sec. 11 we proposed an adjustment to our sponge training algorithm and objective to restore the energy consumption levels of sponge models while keeping their prediction accuracy unaltered. Specifically, we test if fine-tuning the sponge models with Eq. (6.5) we can diminish the effect of a sponge poisoning attack, repairing the model. In Table 6.4 we evaluate the effectiveness of such reversing approach when considering the configurations in Table 6.3 showing the highest energy consumption. Furthermore, in the Appendix we report more results and analysis when varying the hyperparameters and training configurations. The overall results suggest that a model can be restored after being subject to a sponge poisoning attack, but this requires fine-tuning the model for long epochs. If it is fine-tuned only for a few epochs, the model will consume more or will have lower accuracy performance than pristine trained models. However, although this approach can effectively repair the sponge model, training for a huge amount of epochs has the exact cost of training a new model, which is not affordable for all users. Essentially, assuming to work in the outsourced training scenario, seen in Sec. 6.1, the victim has insufficient computational resources, so the reversing approach can not be applied, leaving them exposed to our sponge poisoning attack. In conclusion, once a model has been subjected to a sponge attack, it is more cost-effective for the user to throw it away because retraining it to restore it can be too expensive.

6.3 Concluding Remarks

In this Chapter, we formulated a novel objective function to target energy consumption and derived the first sponge poisoning attack. **Our attack reaches the goal of increasing the victim model's energy consumption and prediction latency by vanishing the effect of DNNs ASIC accelerators, which leverage sparsity in the model's activations to reduce energy consumption and the number of performed mathematical operations.** We initially tested the effectiveness of our attack using the function to approximate the energy consumption proposed in [245]. However, we demonstrated its inadequacy in the poisoning setting. Thus, we proposed utilizing a different objective function, which allows our attack to increase the energy consumption while preserving, or even increasing, the model's prediction accuracy. We further analyzed the models generated by our attack, noting that internal module operators relying on *max* operators (e.g., ReLU and MaxPooling) are more vulnerable to sponge attacks. We then have shown that our attack can be very effective even if the attacker only controls a few model updates during training, making it suitable for targeting federated learning scenarios where attackers usually can compromise only a few nodes. Finally, we investigated the possibility of reversing the influence of sponge poisoning, thereby bringing the models back to a lower energy consumption profile. However, we showed that our sponge poisoning attack is irreversible unless the victim completely retrains the attacked model, thus increasing training costs.

In conclusion, we found a novel security threat, namely sponge poisoning, in outsourcing the training of ML models to untrusted entities. We believe this work may open the door toward designing possible novel defenses and regularization terms for training energy-saving machine learning models. In future works, we aim to extend our approach to a backdoor sponge poisoning attack, where the model increases the energy consumption only for the test samples containing a peculiar pattern, or to go beyond the outsourcing scenario, considering the cases where the attacker has access only to a few training data samples and can not tamper with the learning algorithm.

Chapter 7

Poisoning Clustering Under Limited Knowledge

Research Question #5

Can we poison clustering algorithms under limited knowledge?

Clustering algorithms play a fundamental role as tools for automatic decision-making. Due to the widespread use of these applications, a robustness analysis of this family of algorithms against adversarial noise has become imperative. However, most papers in adversarial machine learning deal with supervised learning models. In this Chapter we expand the knowledge about the security of traditional clustering algorithms used in practice (e.g., K-Means, Spectral, and Hierarchical clustering) by responding to our RQ#5. Therefore, we explore how to stage a poisoning attack against clustering to test their robustness and assume limited knowledge of the attacker, making the attack more practical.

Underlying Problem. A critical challenge we address when staging an attack against clustering algorithms is that most of them are not differentiable; thus, adversarial gradient-based approaches – widely used in supervised settings – are not directly applicable. Since, in general, the machine learning field is currently dominated by gradient-based methods, this may represent a possible reason for the limited interest in this field. In addition, we explore how to stage black-box adversarial attacks against clustering, not only because they can help find common weaknesses of clustering algorithms but also pave the road toward general rules for the formulation of robust clustering algorithms.

Related Work. To the best of our knowledge, however, only a few works have addressed the problem of poisoning clustering algorithms, and only one has investigated how to stage the attack under a black-box threat model.

The problem of devising specific attacks to subvert the clustering algorithm was first brought to light by Dutrisac and Skillicorn [81] and Skillicorn [247]. They pointed out that some samples could be misclustered by positioning them close to the original cluster boundary so that a new *fringe* cluster is formed. Their attack consists of adding points between two clusters to merge them, based on the notion of bridging. Biggio et al. [29] provided the first theoretical formulation for the adversarial clustering problem and proposed a white-box attack to fool single-linkage hierarchical clustering. The attacker is assumed to inject new poisoning points into the target dataset to violate the system availability and deteriorate the clustering results. A similar work has been later proposed in [30], which considers complete-linkage hierarchical clustering. Crussell and Kegelmeyer [68] developed a poisoning algorithm to fool DBSCAN-based algorithms by selecting and then merging arbitrary clusters.

However, all the aforementioned works assume that the attacker has perfect knowledge about the clustering algorithm under attack. In these works, the authors usually leverage the internal behavior of the clustering methods under study to craft *ad-hoc* adversarial noise. The only work assuming to fool clustering in a black-box manner has been developed by [55]. The authors proposed a derivative-free, black-box attack strategy to target clustering algorithms. Their strategy consists in manipulating only one specific input sample feature-by-feature to corrupt the clustering decision boundary. However, their method only applies when working on binary linearly separable tasks.

Contributions and Outline. In an attempt to fill this gap, we developed a gradient-free black-box adversarial attack in [64] to test the robustness of generic clustering algorithms, not only linearly separable ones. We assume that the attacker can only perform queries to it. We formulate in Sec. 7.1 the poisoning clustering problem assumes the attackers can query the target clustering algorithm as a service only, and they are constrained in the maximum amount of noise to inject to avoid detection. In the absence of any derivative information, we perform the optimization with a custom approach, explained in Sec. 7.1, inspired by the Abstract Genetic Algorithm (AGA). We prove in Sec. 10 that the resulting algorithm has significant convergence properties to find the optimal perturbation for multiple samples and features simultaneously. We test the effectiveness of our attack in Sec. 7.2 considering three datasets and multiple clustering methods, showing that our algorithm can significantly affect the clustering performance. Furthermore, we compare our algorithm against [55], showing that our attack can find more powerful poisoning samples with the same limited knowledge assumption. We conclude this Chapter in Sec. 7.3 pointing out the main results of our analysis on the robustness of the clustering algorithm while emphasizing that there is still a long way to go to make this attack practical in more challenging environments.

7.1 Poisoning Clustering via Adversarial Queries

In this section, we introduce the main contribution of our work [64], a black-box gradient-free attack to test the robustness of clustering algorithms. We formulate the problem as a constrained minimization program, general in its structure and customizable by the attacker according to their capability constraints. We propose a genetic algorithm to solve it and investigate its theoretical and empirical convergence properties. Conversely to previous works, we do not assume any information about the internal structure of the victim clustering algorithm, and we allow the attacker to query it as a service only.

7.1.1 Attack Formulation

Notation. We here recap the notation used in the remaining of this Chapter. We further report on page v the list of symbols defined below.

Let $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ denote a feature matrix representing the dataset to be poisoned, where n is the number of samples, and d is the number of features. We define $\mathcal{C} : \mathbb{R}^{n \times d} \mapsto \{1, \ldots, K\}^n$ to be the target clustering algorithm, that separates n samples into K different classes ($1 \leq K \leq n$). We remark that the attacker can retrieve the number of clusters by querying the clustering algorithm, which may also change during the evaluation.

We consider the problem of crafting an adversarial mask $\boldsymbol{\epsilon}$, to be injected into $\boldsymbol{X}$, such that the clustering partitions $\mathcal{C}(\boldsymbol{X})$ and $\mathcal{C}(\boldsymbol{X} + \boldsymbol{\epsilon})$ are different to a certain degree. In real scenarios, the attacker may follow some policies on the nature of the attack, usually imposed by intrinsic constraints on the problem at hand [23]. We model the scenario in which the attacker may want to perturb a specific subset of samples $T \subseteq \{1, \ldots, n\}$, in such a way that the attack is less human-detectable, i.e., by constraining the norm of $\boldsymbol{\epsilon}$ [195]. In our work, the attacker's capability constraints [23] are thus defined by (a) an *attacker's maximum power* δ, which is the maximum amount of noise allowed to be injected in a single entry $\boldsymbol{x}_{ij}$, (b) an *attacker's maximum effort* γ, which is the maximum number of manipulable entries of $\boldsymbol{X}$. Further, we assume the attacker has access to the feature matrix $\boldsymbol{X}$, and she can query the clustering algorithm $\mathcal{C}$ under attack. Similarly to [55] the adversary exercises a causative influence by manipulating part of the data to be clustered without any further information about the victim's algorithm $\mathcal{C}$.

Given these considerations, an optimization program for our task is proposed as follows:

$$\min_{\epsilon \in \mathcal{E}_{T,\delta}} \phi(\mathcal{C}(\boldsymbol{X}), \mathcal{C}(\boldsymbol{X} + \boldsymbol{\epsilon})) \tag{7.1}$$

where ϕ is a similarity measure between clusterings, and

$$\mathcal{E}_{T,\delta} = \{\boldsymbol{v} \in \mathbb{R}^{n \times d}, \|\boldsymbol{v}\|_\infty < \delta \wedge \boldsymbol{v}_i = \boldsymbol{0} \ \forall i \notin T\} \tag{7.2}$$

$\mathcal{E}_{T,\delta}$ is the *adversarial attack space*, which defines the space of all possible adversarial masks that satisfy the maximum power constraints and perturb only the samples in T. A problem without such capability constraints can be denoted with $\mathcal{E}_{\boldsymbol{X},\infty}$. Note that γ is not directly referenced in $\mathcal{E}_{T,\delta}$ but is bounded by T itself, namely $\gamma = |T| \cdot d$.
We further elaborate Eq. (7.1) by searching for low Power & Effort (P&E) noise masks to enforce the non-detectability, or stealthiness, of the attack. To this end, we adopt a similar strategy as in [171], which adds a penalty term $\lambda\|\boldsymbol{\epsilon}\|_p$ to the cost function, usually with $p = 0, 2$ or ∞. Following this approach, we reformulate Eq. (7.1) by including a penalty term that considers both the attacker's P&E, which leverages the ∞ and 0 norms, respectively. The optimization program becomes:

$$\min_{\boldsymbol{\epsilon} \in \mathcal{E}_{T,\delta}} \phi(\mathcal{C}(\boldsymbol{X}), \mathcal{C}(\boldsymbol{X} + \boldsymbol{\epsilon})) + \lambda\|\boldsymbol{\epsilon}\|_0\|\boldsymbol{\epsilon}\|_\infty \tag{7.3}$$

This choice keeps the optimization program interpretable since it establishes a straightforward connection to our minimization desiderata (low P&E). In addition, our penalty term can be seen as a proxy function for $\|\boldsymbol{\epsilon}\|_p$, granting similar regularization properties to the optimization. Indeed the P&E penalty is an upper bound to the single norm term, as the following lemma shows:

Lemma 1. *Let $\boldsymbol{x} \in \mathbb{R}^n$, and $p, q \in \mathbb{R} \cup \{+\infty\}$ such that $1 \leq p \leq q < +\infty$ then:*

$$\|\boldsymbol{x}\|_p \leq \|\boldsymbol{x}\|_0\|\boldsymbol{x}\|_q \tag{7.4}$$

Proof. The case $\boldsymbol{x} = \boldsymbol{0}$ is trivial. Suppose that $\forall i$, $x_i \neq 0$, then for a known result on the equivalence of norms in $\mathbb{R}^n$ [123] we know that $\|\boldsymbol{x}\|_p \leq n^{(1/p-1/q)}\|\boldsymbol{x}\|_q$, thus:

$$\begin{aligned} \|\boldsymbol{x}\|_p &\leq n^{(1/p-1/q)}\|\boldsymbol{x}\|_q \leq n^{1/p}\|\boldsymbol{x}\|_q \leq n\|\boldsymbol{x}\|_q \\ &= \|\boldsymbol{x}\|_0\|\boldsymbol{x}\|_q \end{aligned} \tag{7.5}$$

Suppose now, without loss of generality that $\boldsymbol{x} = (x_1, \ldots, x_m, 0, \ldots, 0)^\top$, such that $\forall i \in \{1, \ldots, m\}$, $x_i \neq 0$. Consider its projection $\boldsymbol{x}'$ onto the axes $1, \ldots, m$, then $\forall p \geq 0$, $\|\boldsymbol{x}\|_p = \|\boldsymbol{x}'\|_p$. Thus Eq. (7.5) holds since:

$$\|\boldsymbol{x}\|_p = \|\boldsymbol{x}'\|_p \leq m\|\boldsymbol{x}'\|_q = \|\boldsymbol{x}\|_0\|\boldsymbol{x}\|_q$$

□

7.1.2 Threat Algorithm

The approach we used to optimize Eq. (7.3) takes its inspiration from Genetic Algorithms (GA) [103]. These methods nicely fit our black-box setting since they do not require any particular property on the function to be optimized. Furthermore, our algorithm possesses solid convergence properties. In Sect. 10 we show that our algorithm is an instance of the *Abstract Genetic Algorithm* (AGA), as presented in [82; 83], and we prove its convergence.

An additional constraint, usually imposed in real-world scenarios, is represented by the limited number of queries performed on the algorithm under attack [5]. Classical approaches in GAs usually create large, fixed-size populations at each generation, and this, in turn, requires computing the fitness score multiple times, querying $\mathcal{C}$ for each individual in the population, thereby making the process query-inefficient. To address this issue, we propose a growing size population approach. We start with a population Θ of size equal to 1 and, generation by generation, we grow it by producing a new individual. To still harness the explorative power of GAs, we use a high mutation rate and allow the population set Θ to grow by keeping track of all the previously computed individuals. In the case of memory-aware applications, our method can be extended by controlling the size of Θ, in particular, by pruning low-fitness candidates. However, in our experiments, we adopted a different technique aimed at speeding up the convergence of the optimization algorithm by reducing the number of generations.

```
Algorithm 6: Black-box poisoning
  Input: $\boldsymbol{X} \in \mathbb{R}^{n \times d}, \mathcal{C}, \delta, T, G, l$
  Output: $\boldsymbol{\epsilon}^*$

Initialize $\boldsymbol{\epsilon}^{(0)} \in \mathcal{E}_{T,\delta}$ randomly
$\Theta = \{\boldsymbol{\epsilon}^{(0)}\}$
for $i$ in $1, \dots, N$ do
    for $g$ in $0, \dots, G-1$ do
        $\boldsymbol{\epsilon}_{ch}^{(g+1)} = \texttt{choice}(\Theta, l)$
        $\boldsymbol{\epsilon}_{cr}^{(g+1)} = \texttt{crossover}(\boldsymbol{\epsilon}^{(g)}, \boldsymbol{\epsilon}_{ch}^{(g+1)})$
        $\boldsymbol{\epsilon}^{(g+1)} = \texttt{mutation}(\boldsymbol{\epsilon}_{cr}^{(g+1)}, \delta, T)$
        $\Theta = \Theta \cup \{\boldsymbol{\epsilon}^{(g+1)}\}$
return $\boldsymbol{\epsilon}^* = \arg\min_{\boldsymbol{\epsilon} \in \Theta} l(\boldsymbol{\epsilon})$
```

Algorithm 6 describes our optimization approach. It takes as input the feature matrix $\boldsymbol{X}$, the clustering algorithm $\mathcal{C}$, the target samples T, the maximum attacker's power δ, the total number of generations G (the attacker's budget in term of queries) and the attacker's objective function l (which in our case is the one defined in Program (7.3)). The resulting output is the optimal adversarial noise mask $\boldsymbol{\epsilon}^*$ that minimizes l. At each generation, a new adversarial mask $\boldsymbol{\epsilon}^{(g+1)}$ is generated and added to a population set Θ containing all previous masks.

The core parts of our optimization process are the stochastic operators – `choice`, `crossover` and `mutation` – that we use for crafting new candidate solutions with a better fitness score. In the following, we describe their implementation.

Choice The choice operator is used to decide which candidates will be chosen to generate offspring. We adopt a roulette wheel approach [103], where only one candidate is selected with a probability proportional to its fitness score, which in turn is *inversely* proportional to the attacker's objective function l. Given a candidate $\boldsymbol{\epsilon}^{(i)}$, its probability to be chosen for the production process $p(\boldsymbol{\epsilon}^{(i)})$ is equal to:

$$p(\boldsymbol{\epsilon}^{(i)}) = \frac{\exp(-l(\boldsymbol{\epsilon}^{(i)}))}{\sum_{\boldsymbol{\epsilon} \in \Theta} \exp(-l(\boldsymbol{\epsilon}))} \tag{7.6}$$

We remark that our choice operator picks just one adversarial noise mask that is then used in the crossover step.

Crossover The crossover operator simulates the reproduction phase by combining different candidate solutions (parents) for generating new ones (offspring). Commonly, crossover operators work with binary-valued strings, however, since our candidates are matrices in $\mathcal{E}_{T,\delta}$, we propose a variant. Given two candidates $\boldsymbol{\epsilon}', \boldsymbol{\epsilon}'' \in \mathcal{E}_{T,\delta}$, the new offspring is generated starting from $\boldsymbol{\epsilon}'$, then with probability equal to p_c each entry i, j is swapped with the entry i, j in $\boldsymbol{\epsilon}''$. The crossover operator has probability p_c of being applied; in the case of failure, $\boldsymbol{\epsilon}'$ itself is chosen as an offspring.

Mutation The mutation is a fundamental operator, usually applied to the offspring generated by the crossover to introduce genetic variation in the current population. Our operator mutates each entry ϵ_{ij} s.t. $i \in T$ with probability p_m by adding an uniformly distributed random noise in the range $[-\delta, \delta]$. The resulting perturbation matrix is subsequently clipped to preserve the constraints dictated by $\mathcal{E}_{T,\delta}$.
Moreover, to enforce the low attacker's effort desiderata, we also perform zero-mutation, meaning that each entry of the mask is set to zero with probability p_z.

Time complexity analysis This section provides a time complexity analysis for Algorithm 6. In step 8, the objective function is computed, requiring, in turn, to execute the clustering algorithm $\mathcal{C}$ with complexity $O(\mathcal{C}(nd))$. Step 9 performs a crossover between two adversarial masks in $O(nd)$ time. The mutation of Step 10 is similarly computed in $O(nd)$ time. The overall time complexity is, thus, given by $O(G(\mathcal{C}(nd) + 2nd)) = O(G\mathcal{C}(nd))$, with G equal to the number of generations. The complexity of the clustering algorithm $\mathcal{C}(nd)$ is a critical point in the efficiency of the attack. As an example, considering K-Means, we have a polynomial-time of $O(G(ndKt + 2n)) = O(GndKt)$, with K being the number of clusters and t the number of iterations for the clustering algorithm.

Speeding up the convergence By just generating a new individual at each generation, our proposed method has the major drawback of being slow at converging. To counter this problem, inspired by the work of [105], we decided to "imprint" a direction to the generated adversary mask to move the adversarial samples toward the target cluster. Since we lack the gradient information, the centroids information is leveraged instead. We propose the following approach: each adversarial mask $\boldsymbol{\epsilon} \in \mathcal{E}_{T,\delta}$ is generated with the additional constraint that $\forall i, j \; \epsilon_{ij} \geq 0$. After this, the mask is multiplied by a *direction matrix* $\boldsymbol{\psi}$ with $\psi_{ij} = \text{sgn}(c_j^{(t)} - c_j^{(v)})$, $\boldsymbol{c}^{(t)}$ and $\boldsymbol{c}^{(v)}$ being respectively the target and victim cluster centroids estimated from the victim data. The estimation is performed by averaging the samples in the corresponding cluster. This variant can be easily implemented by changing the initialization of $\boldsymbol{\epsilon}^{(0)}$ and the mutation step only. It follows that the resulting adversarial attack space is now reduced to $\mathcal{E}'_{T,\delta} \subset \mathcal{E}_{T,\delta}$. We still grant that the capability constraints are respected, and the convergence properties hold, although the quality of the found optimum may be inferior. In addition, we noticed that, without using this strategy, the optimization algorithm was more sensitive to the choice of hyper-parameters. Therefore, we have decided to adopt this strategy, which makes our algorithm more efficient and less sensitive to the choice of hyperparameters.

7.1.3 Convergence properties

In general, GAs do not guarantee any convergence property [228]. However, under some more restrictive assumptions, it can be shown that they converge to an optimum. In this section, we show that our algorithm can be thought of as an instance of the Abstract Genetic Algorithm (AGA) as presented in [82; 83]. Subsequently, we give proof of convergence. We report in Algorithm 7 the generic pseudo-code for AGAs. In [82; 83], the authors show that methods such as classical Genetic Algorithms and Simulated Annealing can be thought of as instances of Algorithm 7. Further, they prove their probabilistic convergence to a (global) optimum. Following the same theoretical framework, we show that our algorithm satisfies all the convergence conditions. Before doing so, we present the framework and adapt our algorithm to comply with it.
Let S be a set of candidates, and S^* be a set of finite lists over S, representing all the possible finite populations. A *neighborhood function* is a function $N : S \mapsto S^*$ that assigns neighbors to each individual in S. A *parent-list*, is a list of candidates able to generate offspring, with

Algorithm 7: Abstract Genetic Algorithm (pseudo-code)

```
Make initial population
while not stopping condition do
    Choose parents from population
    Let the selected parents Produce children
    Extend the population by adding children to it
    Select elements of the extended population to survive for the next cycle
return the optimum of the population
```

Algorithm 8: Abstract Genetic Algorithm.

```
Input: $S^*$
Output: $x$

Create an $x \in S^*$
while not stopping condition do
    Draw $\alpha$, $\beta$ and $\gamma$
    $q = f_c(\alpha, x)$
    $y = \bigcup_{z \in q} f_p(\beta, z)$
    $x' = x \cup y$
    $x = f_s(\gamma, x')$
return output the actual population $x$
```

$P \subseteq S^*$ denoting the set of all parent-list. In our algorithm, a population is represented by a list $[\boldsymbol{\epsilon}^{(0)}, \dots, \boldsymbol{\epsilon}^{(g)}]$, therefore $S = \mathcal{E}_{T,\delta}$, $S^* = \mathcal{E}^*_{T,\delta}$, $P = \{[\boldsymbol{\epsilon}^{(i)}, \boldsymbol{\epsilon}^{(j)}] \mid \boldsymbol{\epsilon}^{(i)}, \boldsymbol{\epsilon}^{(j)} \in \mathcal{E}_{T,\delta}\}$.

Let $f : X \mapsto Y$ be a function belonging to $\mathcal{F}$, the set of all functions from X to Y. Further, let $(\Omega, \mathcal{A}, \mathbb{P})$ be a probability space and $g : \Omega \mapsto \mathcal{F}$ be random variable. We define the *randomized* f to be the function $f(\omega, x) = g(\omega)(x)$. Following this definition and [82], Algorithm 7 can be then detailed as reported in Algorithm 8.

with $f_c : A \times S^* \mapsto \mathcal{P}(P)$ being the *choice function*, $f_p : B \times P \mapsto \mathcal{P}(S)$ being the *production function* and $f_s : C \times S^* \mapsto S^*$ being the *selection function*. In our case, we define:

1. $f_c(\alpha, x) = \{[\texttt{choice}(\alpha, x), x_{-1}]\},\ \forall \alpha \in A$
2. $f_p(\beta, [s_1, s_2]) = \texttt{mutation}(\beta, \texttt{crossover}(\beta, s_1, s_2))),\ \forall \beta \in B$
3. $f_s(\gamma, x') = x'$ (Note that our selection is deterministic)

Where x_{-1} is the most recent candidate in the population. In the above pseudo-code, we have explicitly stated the randomization of our procedures `choice`, `mutation`, `crossover` for clarity. The stochastic processes regulating the drawings of α, β, and γ always maintain the same distributions regardless of the current generation, meaning that the probability of generating a new population x_{new} from another one x_{old} does not change over the generations.

We now introduce and extend some definitions presented in [82]:

1. A neighborhood structure is *connective* if: $\forall s \in S, \forall t \in S : s \mapsto t$, where $\mapsto$ stands for the transitive closure of the relation $\{(s, t) \in S \times S \mid t \in N(s)\}$.
2. A choice function is *generous* if: (a) $\{[s, t] \mid s, t \in S\} \subseteq P$ and (b) $\forall x \in S^*$, $\forall s_1, s_2 \in x : \mathbb{P}([s_1, s_2] \in f_c(\alpha, x)) > 0$.
3. A production function is *generous* if: $\forall s_1, s_2 \in S$, $\forall t \in N(s_1) \cup N(s_2) : \mathbb{P}(t \in f_p(\beta, [s_1, s_2])) > 0$.
4. A selection function is *generous* if: $\forall x \in S^*$, $\forall s \in x : \mathbb{P}(s \in f_s(\gamma, x)) > 0$.

5. A selection function is *conservative* if: $M_x \cap f_s(\gamma, x) \neq \emptyset$, with $M_x = \{s \in x \mid \forall t \in x : f(s) \leq f(t)\}$.

In [82], the authors further make a little technical assumption about the sets A, B, and C, requiring them to be countable, with positive probability for all their members. This is easily achieved in real applications considering the finiteness of the floating point representations.
Now we are ready to prove the following theorem:

Theorem 7.1.1. *Algorithm 8 almost surely reaches a global optimum.*

Proof. Given the previous considerations, the following statements hold:

1. *Our neighborhood structure is connective*: by the definition of our mutation operator, it holds that $N(\boldsymbol{\epsilon}^{(i)}) = \mathcal{E}_{T,\delta}, \forall \boldsymbol{\epsilon}^{(i)} \in \mathcal{E}_{T,\delta}$.

2. *Our choice function is generous*: this follows from (a) the definition of P, and from (b) the positivity of the softmax function in Equation 6.

3. *Our production function is generous*: See point 1.

4. *Our selection function is generous*: we allow all the candidates to survive with probability 1.

5. *Our selection function is conservative*: see point 4.

The proof then follows from Theorem 3 in [83], adjusting the generousness definitions with our versions presented above. The globality of the optimum comes from the fact that our algorithm performs a global search instead of a local one. □

The same conclusions can be drawn for the speed-up heuristic by just replacing each instance of $\mathcal{E}_{T,\delta}$, with $\mathcal{E}'_{T,\delta}$.

7.2 Experimental Analysis

This section presents an experimental evaluation of the proposed attack methodology. We initially present our experimental setup and test the robustness of different clustering algorithms against our poisoning attack. We then test the effectiveness of our attack when further limiting the attacker's knowledge and evaluate its empirical convergence. Finally, we compare our poisoning algorithm with the state-of-the-art-attack [55], highlighting the superior performance of our attack in terms of degradation of clustering algorithm accuracy and minimum attacker effort. We repeated the following experiments five times, reporting the mean with the standard error. The code has been written in PyTorch [85], and it is available at [1].

7.2.1 Experimental Setup

Datasets. We ran the experiments on three datasets, i.e., FashionMNIST [295], 20Newsgroups [155], and CIFAR10 [147]. The FashionMNIST contains 70 000 grayscale images of size 28 × 28 pixels [295]. It is a more challenging version of the MNIST dataset. The 20 Newsgroups is a dataset commonly used for text classification and clustering, which contains 20 000 newspaper articles divided into 20 categories. We then applied a combination of TF-IDF [231] and LSA [154] to embed features into a lower dimensional space. The resulting feature representation has dimension $1\,400 \times 80$. For CIFAR10, we used a ResNet50 for feature extraction and performed clustering on the resulting feature space, obtaining better initial results.

---

[1] https://github.com/Cinofix/poisoning-clustering

Attack Setup. We focused our analysis on both binary and multiple-way clustering problems. We simulated the former scenario with FashionMNIST and 20 Newsgroups allowing the attacker to perturb samples of one victim cluster C_v towards a target cluster C_t. From FashionMNIST we randomly sampled 800 images for class `Ankle boot` (victim cluster) and 800 for class `Shirt` (target cluster). From the 20 Newsgroups dataset, we selected two highly unrelated categories of news, `rec.sport.baseball` (victim cluster) and `talk.politics.guns` (target cluster). We then simulated the latter scenario with CIFAR10. The attacker moves samples from multiple victim clusters towards a target one by simply running our algorithm multiple times with a different victim cluster for each run. We randomly sampled 1 600 images from classes `airplane, frog` and `automobile`. We addressed the multi-way scenario by first moving samples from `airplane` and then from `frog`, always towards the target cluster `automobile`.
In the experiments, we chose a set of target samples T to contain the $s|C_v|$ nearest neighbors belonging to the currently chosen victim cluster, with respect to the centroid of the target cluster. In particular, for FashionMNIST we used 20 different values for s and δ, in the intervals $[0.01, 0.6]$ and $[0.05, 1]$ respectively; for CIFAR10 we used 20 different values for s and δ, in the intervals $[0.01, 0.6]$ and $[0.01, 1.5]$ respectively; for 20 Newsgroups we used 15 different values for s and δ, in the intervals $[0.01, 0.3]$ and $[0.001, 0.3]$ respectively.

Clustering Algorithms. We tested the robustness of three standard clustering algorithms: hierarchical clustering with Ward's criterion [279], K-Means++ [6], and the normalized spectral clustering [243] as presented in [274], with the Local Scaling similarity measure [306]. We further test with the 20 Newsgroups dataset the effectiveness of our attack against two ensemble clustering algorithms, derived from K-Means and spectral clustering algorithms [2]. The two algorithms use the Silhouette value [226], and the clustering with the maximum silhouette score is selected as the best one. In particular, we ran 20 instances of the K-Means algorithm with random centroids initializations, while, for spectral clustering, we ran 3 instances of the algorithm proposed in [274] with 3 different similarity measures. Given a sample pair $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, the measures are:

$$s_{ij} = \frac{\boldsymbol{x}_i^\top \boldsymbol{x}_j}{\|\boldsymbol{x}_i\|_2 \|\boldsymbol{x}_j\|_2} \tag{7.7}$$

$$s_{ij} = \frac{(\boldsymbol{x}_i - \bar{\boldsymbol{x}})^\top (\boldsymbol{x}_j - \bar{\boldsymbol{x}})}{\|\boldsymbol{x}_i - \bar{\boldsymbol{x}}\|_2 \|\boldsymbol{x}_j - \bar{\boldsymbol{x}}\|_2} \tag{7.8}$$

$$s_{ij} = d_{max} - \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2 \tag{7.9}$$

Eq. (7.7) represents the cosine similarity between two samples $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. Eq. (7.8) is the Pearson correlation coefficient, with $\bar{\boldsymbol{x}}$ being the sample mean. Moreover, we introduced a sparsification technique, clamping to 0 all negative values, which improved the clustering performance. Finally, in Eq. (7.9) we define $d_{max} = \max_{ij} \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2$.

Hyperparameters. When running the attack with Alg. 6 we set $\lambda = \frac{1}{\alpha \cdot n \cdot d}$ with $\alpha = 255$ as penalty term for our cost function. In addition, we set the probability of having crossover $p_c = 0.85$, mutation $p_m = 0.05$ and zero-mutation $p_z = 0.001$. The total number of generations, which correspond to the number of queries, was always set to 110, using the speeding-up heuristic proposed in Sect. 7.1.

Performance Metrics. In Eq. (7.3), we indicate with ϕ a function for measuring the similarity between two clustering partitions. In the literature, we can find several metrics used for the evaluation of clusterings [132; 254; 192; 206]; in addition, [29] proposed to adopt the following measure for the evaluation: $d(\boldsymbol{Y}, \boldsymbol{Y}') = \|\boldsymbol{Y}\boldsymbol{Y}^\top - \boldsymbol{Y}'\boldsymbol{Y}'^\top\|_F$, where $\|\cdot\|_F$ is the Frobenius norm, and $\boldsymbol{Y}, \boldsymbol{Y}' \in \mathbb{R}^{n \times k}$ are one-hot encodings of the clusterings $\mathcal{C}(\boldsymbol{X})$ and $\mathcal{C}(\boldsymbol{X} + \boldsymbol{\epsilon})$ respectively. In our

---

[2] hierarchical clustering was not giving good enough clustering performance

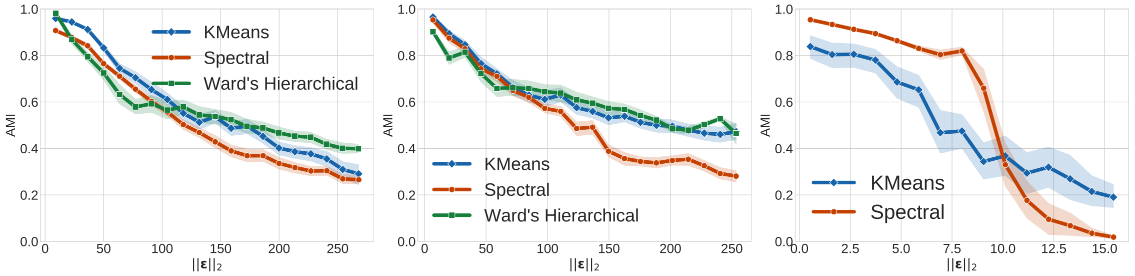


Fig. 7.1: *Robustness analysis with FashionMNIST (left), CIFAR10 (middle), 20 Newsgroups (right). The plots depict the decay of AMI by adversarially perturbing the datasets with increasing noise.*

work, we decided to use the Adjusted Mutual Information (AMI) Score, proposed in [206] since it makes no assumptions about the cluster structure and, as highlighted in [224], it works well even in the presence of unbalanced clusters. Indeed, the clustering partition over the poisoned dataset might also create unbalanced clusters, especially if the attacker wants to move samples only from one to the other. The AMI score between two clustering partitions U and V is given by:

$$AMI(U,V) = \frac{MI(U,V) - \mathbb{E}[MI(U,V)]}{\max\{H(U), H(V)\} - \mathbb{E}[MI(U,V)]} \tag{7.10}$$

where $MI(U,V)$ measures the mutual information shared by the two partitions, $\mathbb{E}[MI(U,V)]$ represents its expected mutual information, and $\max\{H(U), H(V)\}$ is the maximum between the two entropies, which is an upper bound for $MI(U,V)$. AMI is equal to 1 when the two clustering partitions are identical and 0 when they are independent, that is, sharing no information about each other.

7.2.2 Experimental Results

Robustness Analysis. We here test the robustness of chosen clustering algorithms against our poisoning attack.

`FashionMNIST`. We first consider the problem of attacking a binary clustering algorithm with FashionMNIST. In Fig. 7.1 (left), we report the obtained results. We observe that the three algorithms have similar behavior and their clustering accuracy consistently decreases with the increment of the adversarial noise level. In this case, K-Means++ shows better performance than spectral clustering. Therefore the spectral embedding of data samples seems less robust than raw features only. This fact may suggest that some embedding procedures devised for improving clustering accuracy do not necessarily guarantee robustness against adversarial attacks. However, we reserve further discussion on this in future work.

`CIFAR10`. We here consider the performance of our attack when attacking multiple clusters simultaneously with the CIFAR10 dataset. In Fig. 7.1 (middle), we show the performance of the three clustering algorithms under adversarial manipulations. We observe that our attacks significantly decrease the clustering quality for the three algorithms. Even if the ResNet50 features allow cluster algorithms to perform better, they are still vulnerable to adversarial noise. Further, note how the gap in performance of spectral clustering and K-Means++ has even increased when adopting a DNN-generated embedding. In Fig. 7.2, we provide a visual representation of poisoning samples for CIFAR10. We reconstructed the poisoning samples from the feature space using the feature collision strategy adopted in [238], where the target is precisely our poisoning sample.

`20 Newsgroups`. We here report the results of our attack against ensemble clustering algorithms on the 20 Newsgroups dataset. Fig. 7.1 (right) reports the performance of two clustering algorithms under adversarial manipulation. Ensemble methods are known to be more robust against random noise with respect to the expected behavior of the corresponding algorithms [208; 203]; however, our attacking model was able to fool them and significantly decreased their clustering performance.

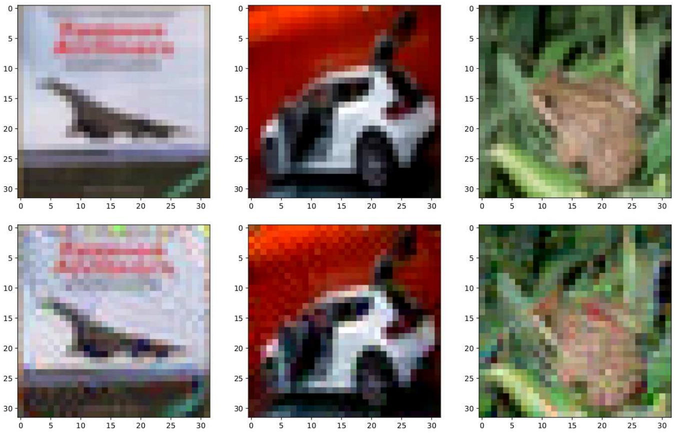


Fig. 7.2: *(Top row) clean samples from CIFAR10. (Bottom row) the corresponding poisoning samples with* $\delta = 0.1$.

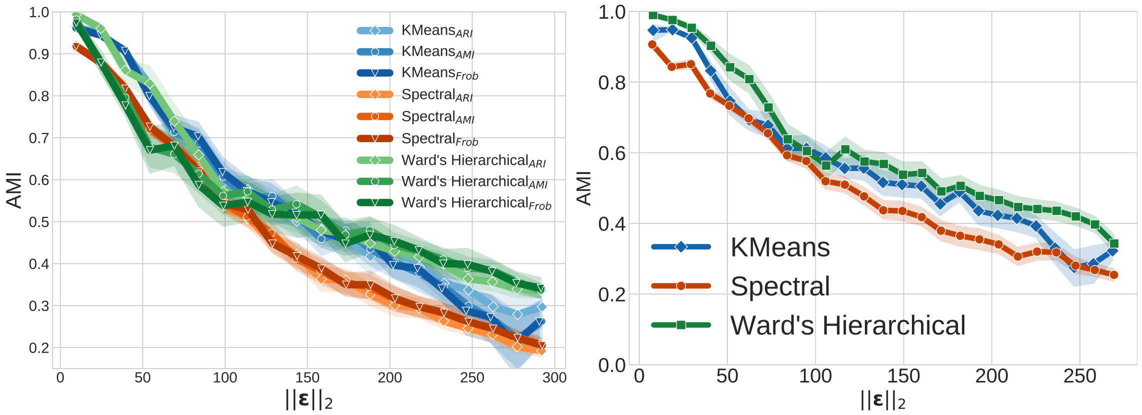


Fig. 7.3: *Robustness analysis on FashionMNIST by changing the similarity measure* ϕ *(left) and crafting the poisoning samples employing a surrogate dataset (right).*

In this case, in a low noise regime, spectral clustering seems to benefit the ensembling technique. However, its behavior follows previous experiments.

Choice of the cost function. In the above presented experimental results, we employed the AMI metrics as the cost function ϕ of our attack formulation presented in Eq. 7.3. We decided to analyze the impact of the clustering similarity function ϕ with FashionMNIST and see if there were significant differences among each other. In Fig. 7.3 (left), shows the variation of results with ϕ equals to ARI, AMI and the negated distance proposed in [29] (referred as "Frob"). The plot shows no substantial difference among the choices of ϕ, suggesting that this hyperparameter does not significantly impact the optimization process.

Surrogate data. We ran additional experiments on a more challenging scenario by relaxing the knowledge assumption of the target data. In this scenario, the attacker does not have access to the target data and can only sample a surrogate dataset from the same distribution. To this end, we have randomly sampled two more subsets of $1,600$ images from FashionMNIST. We use the first subset to create the poisoning samples and then evaluate their effectiveness on the other one. Fig. 7.3 (right) shows that our attack is strong enough to decrease the clustering performance even when the attacker has no access to the target dataset. This means that our attack can transfer to other unknown dataset samples from the same underlying distribution.

Empirical convergence. In addition to the theoretical convergence properties proved in Sect. 10, we propose an empirical analysis of convergence. In particular, for a pre-set configuration of δ and s, we performed a series of attacks on the FashionMNIST dataset, with an increasing number of generations/queries, evaluating the trend of the objective function presented Eq. (7.3). The results are reported in Fig. 7.4. Our algorithm requires a relatively low number of queries to converge to a minimum, except for K-Means++, which presents a slower convergence than spectral clustering, most probably due to the nature of the feature embeddings used.

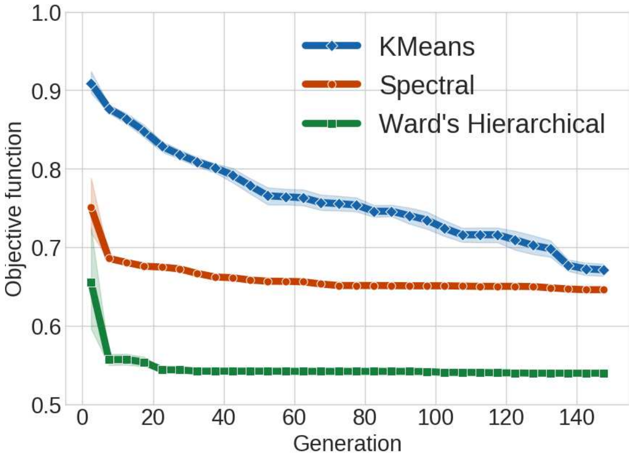


Fig. 7.4: *Convergence of objective function on FashionMNIST.* $\delta = 0.2$, $s = 0.25$.

7.2.3 Experimental Comparison

To our knowledge, the only work dealing with adversarial clustering in a black-box way is [55]. In this work, the authors presented a new type of attack called *spill-over*, in which the attacker wants to assign as many samples as possible to a wrong cluster by poisoning just one of them. They proposed a threat model against linearly separable clusters to generate such kind of adversarial noise.

To have a fair comparison, we performed *spill-over* attacks on the same settings of the aforementioned work, comparing the performance on MNIST and UCI Handwritten Digits datasets[3] [4], targeting Ward's hierarchical clustering algorithm. Further details can be found in Appendix C.

For MNIST, we considered the digit pairs 4&1 and 3&2, while for Digits, we considered the digit pairs 4&1 and 8&9. Our algorithm was run with the $\delta = \Delta$ which is the maximum acceptable noise threshold found by the authors, with $\delta = \Delta/2$ and with $\delta = \infty$. We found the value of Δ used in [55] by looking at the source code. We imposed to attack just one sample ($|T| = 1$), namely the nearest neighbor to the centroid of the target cluster. We performed our experiments 20 times, reporting mean and std values. The results are presented in Table 7.1-7.4 along with more details on the experiments. Although our algorithm achieves its best performance by moving more samples at once, we were able to match, or even exceed, the number of spill-over samples (`#Mis-clust`) achieved in [55], even when halving the attacker's maximum power proposed by the authors. Moreover, the results also show that we could craft adversarial noise masks $\boldsymbol{\epsilon}$, which were significantly less detectable in terms of ℓ_0, ℓ_∞.

In Table 7.5-7.6, we report a comparison for the K-means++ algorithm with UCI Wheat Seeds [43] and MoCap Hand Postures [99] dataset, repeating the same experimental setting of [55]. We obtain the same number of spill-over samples (`#Mis-Clust`) with significant lower Power & Effort. The reader can find further experiments of comparison in the Appendix C.

Finally, in Fig. 7.5 and 7.6, we show a qualitative assessment of the crafted adversarial spill-over samples. Note that the crafted adversarial examples of [55] do not preserve box-constraints

---

[3] A dataset containing 5 620 grayscale images of size 8×8, with intensities in the range $[0, 16]$

METHOD	$\|\epsilon\|_0$	$\|\epsilon\|_2$	$\|\epsilon\|_\infty$	#MIS-CLUST
SPILL-OVER	413	872.8	146.8	2
SPILL-OVER$_{clamp}$	412	828.2	146.8	2
OURS ($\delta = 73.43$)	151 ± 19.2	551.9 ± 36.3	73.4 ± 1.4	12.0 ± 0.0
OURS ($\delta = 146.87$)	30 ± 7.6	479.1 ± 62.5	145.0 ± 3.7	12.0 ± 0.0
OURS ($\delta = \infty$)	29 ± 13.8	757.1 ± 203.2	246.25 ± 16.8	14.3 ± 2.4

Table 7.1: *Comparison on MNIST with digits 3&2.*

commonly adopted for image data. Indeed, pixel intensities exceed 255 and 16 for MNIST and Digits, respectively. We also evaluated the performance of [55] by clamping the resulting adversarial examples (`Spill-over`$_{clamp}$), and we observed that the number of spill-over samples is reduced.

METHOD	$\|\epsilon\|_0$	$\|\epsilon\|_2$	$\|\epsilon\|_\infty$	#MIS-CLUST
SPILL-OVER	152	585.3	159.7	11
SPILL-OVER$_{clamp}$	151	463.2	131.7	9
OURS ($\delta = 79.86$)	117 ± 7.8	528.4 ± 30.1	79.8 ± 2.8	9.1 ± 0.4
OURS ($\delta = 159.72$)	75 ± 22.4	782.7 ± 124.2	159.3 ± 1.3	12.0 ± 4.5
OURS ($\delta = \infty$)	46 ± 19.4	902.7 ± 205	248.3 ± 8.8	14.6 ± 4.5

Table 7.2: *Comparison on MNIST with digits 4&1.*

In conclusion, [55] aims to find Δ, which does not lead to the attack being considered an outlier using the COMD measure at the expense of existing box-constraints. Whereas our purpose consists of proposing an algorithm that can effectively corrupt a black-box clustering algorithm's performance by minimizing the attacker's power and effort (P&E). Indeed, our attacks, as shown in Table 7.3-7.6, show lower ℓ_0 and ℓ_∞ compared to the attack obtained with [55]. These results suggest that our algorithm can craft effective poisoning attacks, even stronger than [55], with less P&E and satisfying the box-constraints.

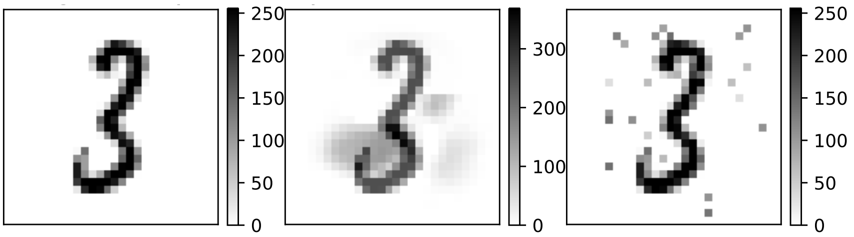


Fig. 7.5: *Spill-over samples for MNIST. The target sample (left), the corresponding adversarial sample crafted with the attack proposed in [55] (middle), and our adversarial sample with $\delta = 146.87$ (right).*

Ablation study. We provide an ablation study for the mutation and zero rate parameters, respectively p_m and p_z, keeping the crossover rate p_c set to 0.85 for the two pairs of MNIST digits. The high crossover rate is a common choice in Genetic Algorithms [116]. We use the same setting described in Sect. 7.2 with datasets MNIST 3&2 and MNIST 4&1.

Fig. 7.7 reveals how the two hyperparameters affect the attacker's effort and the attack efficacy. An increment of the zero rates implies attacks with less effort, while an increment of the mutation inverts this tendency, thus generating more powerful attacks. From Fig. 7.7(left matrices), we

Method	$\|\epsilon\|_0$	$\|\epsilon\|_2$	$\|\epsilon\|_\infty$	#Mis-clust
Spill-over	54	15.70	9.44	21
Spill-over$_{clamp}$	54	15.70	9.44	21
Ours ($\delta = 4.72$)	12 ± 1.20	11.49 ± 1.25	4.7 ± 0	21 ± 0.0
Ours ($\delta = 9.44$)	7 ± 2.85	13.86 ± 2.96	8.12 ± 1.24	21 ± 0.0
Ours ($\delta = \infty$)	4 ± 1.74	15.18 ± 3.16	10.94 ± 1.49	21 ± 0.0

Table 7.3: *Comparison on Digits with digits 8&9.*

Method	$\|\epsilon\|_0$	$\|\epsilon\|_2$	$\|\epsilon\|_\infty$	#Mis-clust
Spill-over	14	23.93	11.89	24
Spill-over$_{clamp}$	11	16.28	9.93	21
Ours ($\delta = 5.94$)	13 ± 1.70	16.27 ± 1.20	5.94 ± 0.0	24 ± 0.0
Ours ($\delta = 11.89$)	7 ± 2.03	19.84 ± 1.96	11.13 ± 0.79	24 ± 0.0
Ours ($\delta = \infty$)	7 ± 2.36	21.06 ± 2.36	12.79 ± 4.34	24 ± 0.0

Table 7.4: *Comparison on Digits with digits 4&1.*

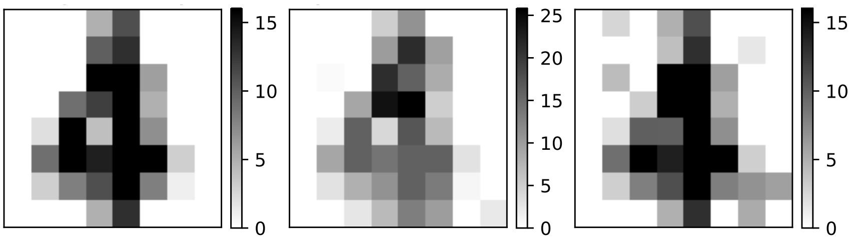


Fig. 7.6: *Spill-over samples for Digits. The target sample (left), the corresponding adversarial sample crafted with the attack proposed in [55] (middle), and our adversarial sample with $\delta = 11.89$ (right).*

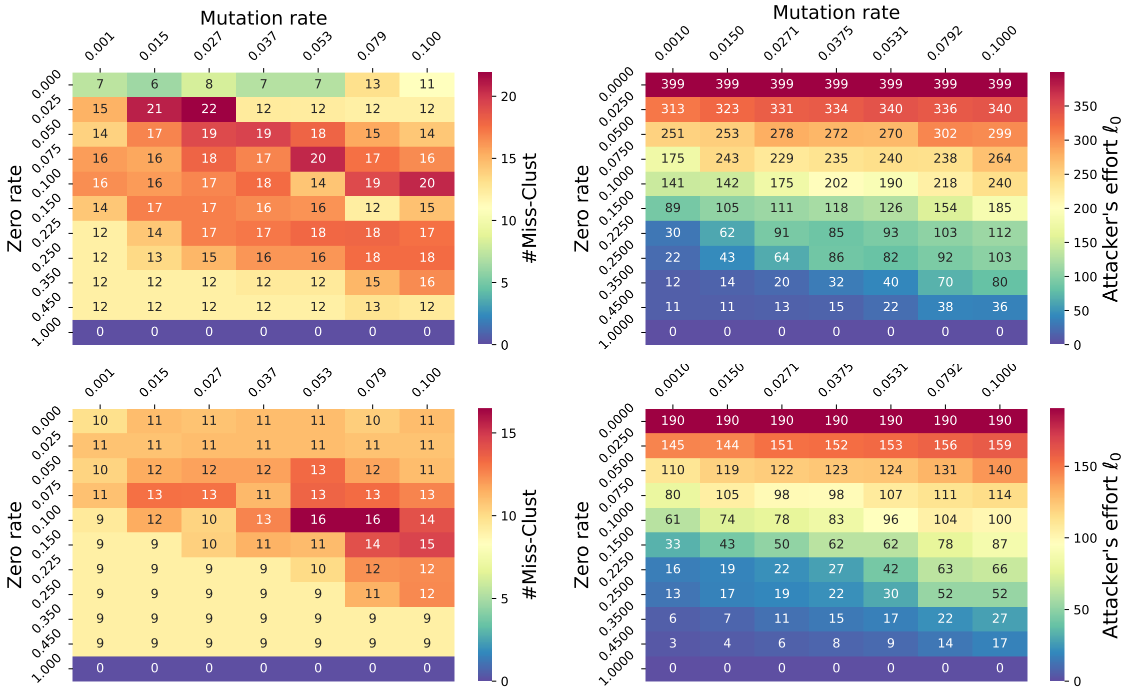
observe that the yellow regions at the bottom left define a good compromise between having a small effort and good attack effectiveness. However, by increasing the attacker's effort (right matrices), we can generate even more effective attacks. The results of the ablation study with the two pairs of digits are similar, suggesting that the same set of hyperparameters can be chosen without significant differences, as also suggested by the wide bottom-left regions of Fig. 7.7 (left matrices) where the number of miss-clustered observations is constant. This behavior suggests that we do not need an extensive hyperparameter tuning procedure to obtain effective poisoning attacks. The results obtained against [55] use a combination of hyperparameters that allows the attacker's effort to be kept low while still maintaining outstanding comparison results. However, a better choice of hyperparameters would have allowed us to further improve our results in terms of miss-clustered points and the attacker's effort.

METHOD	$\|\boldsymbol{\epsilon}\|_0$	$\|\boldsymbol{\epsilon}\|_2$	$\|\boldsymbol{\epsilon}\|_\infty$	#MIS-CLUST
SPILL-OVER	7	0.42	0.30	2
OURS ($\delta = 0.15$)	3 ± 0.79	0.14 ± 0.04	0.10 ± 0.03	2.0 ± 0.0
OURS ($\delta = 0.30$)	3 ± 0.76	0.28 ± 0.09	0.21 ± 0.06	2.0 ± 0.0

Table 7.5: *Comparison for Seeds.*

METHOD	$\|\boldsymbol{\epsilon}\|_0$	$\|\boldsymbol{\epsilon}\|_2$	$\|\boldsymbol{\epsilon}\|_\infty$	#MIS-CLUST
SPILL-OVER	9	44.42	20.0	5
OURS ($\delta = 10$)	1 ± 0.48	5.13 ± 1.86	5.0 ± 1.86	5.0 ± 0.0
OURS ($\delta = 20$)	1 ± 1.14	8.50 ± 6.61	7.74 ± 5.18	5.0 ± 0.0

Table 7.6: *Comparison for MoCap Hand Postures.*



	0.001	0.015	0.027	0.037	0.053	0.079	0.100
0.000	7	6	8	7	7	13	11
0.025	15	21	22	12	12	12	12
0.050	14	17	19	19	18	15	14
0.075	16	16	18	17	20	17	16
0.100	16	16	17	18	14	19	20
0.150	14	17	17	16	16	12	15
0.225	12	14	17	17	18	18	17
0.250	12	13	15	16	16	18	18
0.350	12	12	12	12	12	15	16
0.450	12	12	12	12	12	13	12
1.000	0	0	0	0	0	0	0

	0.0010	0.0150	0.0271	0.0375	0.0531	0.0792	0.1000
0.0000	399	399	399	399	399	399	399
0.0250	313	323	331	334	340	336	340
0.0500	251	253	278	272	270	302	299
0.0750	175	243	229	235	240	238	264
0.1000	141	142	175	202	190	218	240
0.1500	89	105	111	118	126	154	185
0.2250	30	62	91	85	93	103	112
0.2500	22	43	64	86	82	92	103
0.3500	12	14	20	32	40	70	80
0.4500	11	11	13	15	22	38	36
1.0000	0	0	0	0	0	0	0

	0.001	0.015	0.027	0.037	0.053	0.079	0.100
0.000	10	11	11	11	11	10	11
0.025	11	11	11	11	11	11	11
0.050	10	12	12	12	13	12	11
0.075	11	13	13	11	13	13	13
0.100	9	12	10	13	16	16	14
0.150	9	9	10	11	11	14	15
0.225	9	9	9	9	10	12	12
0.250	9	9	9	9	9	11	12
0.350	9	9	9	9	9	9	9
0.450	9	9	9	9	9	9	9
1.000	0	0	0	0	0	0	0

	0.0010	0.0150	0.0271	0.0375	0.0531	0.0792	0.1000
0.0000	190	190	190	190	190	190	190
0.0250	145	144	151	152	153	156	159
0.0500	110	119	122	123	124	131	140
0.0750	80	105	98	98	107	111	114
0.1000	61	74	78	83	96	104	100
0.1500	33	43	50	62	62	78	87
0.2250	16	19	22	27	42	63	66
0.2500	13	17	19	22	30	52	52
0.3500	6	7	11	15	17	22	27
0.4500	3	4	6	8	9	14	17
1.0000	0	0	0	0	0	0	0

Fig. 7.7: *Ablation study for MNIST 3&2 (top row) and MNIST 4&1 (bottom row). For each row, the two matrices report the number of miss clustered samples (left) and the number of manipulated pixels (right) when varying the mutation rate and zero rate.*

7.3 Concluding Remarks

In this Chapter, we investigated the feasibility of staging a poisoning attack against a clustering algorithm with limited knowledge. For this purpose, we examined our black-box adversarial algorithm proposed in [64]. We assumed the attack to have only the capacity to query the victim clustering algorithm and check for output variations. In the absence of any gradient information, we developed a genetic black-box algorithm, as done in [5], to optimize the poisoning noise injected in the victim data to mislead the clustering algorithm. We have conducted several experiments to test the robustness of classical single and ensemble clustering algorithms on different datasets, showing that they are vulnerable to our attack. We have further compared our method with a state-of-the-art black-box poisoning strategy, showing that we outperform their results in terms of attacker's capability requirements and misclustering error. Finally, we have proven our algorithm's empirical and theoretical convergence towards the optimum poisoning noise.

In conclusion, responding to our research question, "Can we poison clustering algorithms under limited knowledge?", we would say "**yes, poisoning attacks can be staged against clustering algorithms even when the attacker has only query (or oracle) access to the victim clustering algorithm**". Unfortunately, the drawback of this attack is that it requires many queries, burdening the computational time required to run the attack. Each objective evaluation requires executing the clustering algorithm on the novel perturbation, which may not be feasible when considering computational demanding clustering algorithms or when scaling to large datasets. It is thus demanding under these scenarios to find more novel heuristics for decreasing the number of queries, or reducing the computational costs of clustering algorithms, for example, by approximating them. We believe this work can pave the road toward developing novel poisoning attacks against clustering algorithms to assess their robustness in the presence of oracle attackers.

Chapter 8

Open Challenges and Conclusions

Research Question #6

Which are the open challenges in the poisoning ML literature?

In this Chapter we outline in Sec. 8.1 these challenges and explore in Sec. 8.2 promising future research directions to tackle them. We finally summarize in Sec. 8.3 the main contributions of this thesis, along with the main results found with the analysis, their limitations, and future plans to address them.

8.1 Current Limitations

Although data poisoning has been widely studied in recent years, from our literature review, we have found that two main challenges are still hindering their thorough development.The historical development of the three types of attacks illustrated in Chapter 3 (Sec. 3.3.5) highlights how recent work aims to solve or mitigate two key challenges (i) considering more realistic threat models, and (ii) designing more effective and scalable poisoning attacks.

8.1.1 Unrealistic Threat Models

The first challenge we formulate here questions some of the threat models considered in previous work. The reason is that such threat models are not well representative of what may happen in many real-world application settings. For example, Fowl et al. [94] and Feng et al. [93] assume that the attacker controls almost the entire training dataset to mount an indiscriminate poisoning attack against DNNs effectively. While this may happen in certain hypothesized situations, it is also not surprising that a poisoning attack works if the attacker controls a considerable fraction of the training set. We believe that poisoning attacks should be considered a realistic threat only when it is assumed that the attacker can control a small fraction of the training points. We refer the reader to a similar discussion in the context of poisoning federated learning in [241].

Another limitation of threat models considered for poisoning attacks is that, in some cases, exact knowledge of the *test* samples is implicitly assumed. For example, [238] and [101] optimize a targeted poisoning attack to induce misclassification of few specific *test* samples. In particular, the attack is optimized and tested using the same *test* samples, different from work which optimizes the poisoning points using validation data and then tests the attack impact on a separate test set [27; 197]. This evaluation setting enables the attack to reach higher success rates. However, at the same time, there is no guarantee that the attack will generalize even to minor variations of the considered test samples, questioning its applicability outside of settings in which the attacker has exact knowledge of the test inputs. For instance, the attack may not work as expected in physical domains, where cameras acquire images under varying illumination and environmental conditions.

In such cases, it is clear that the attacker can not know the specific realization of the test sample beforehand, as they do not control the acquisition conditions.

On a similar note, only a few studies on backdoor poisoning have considered real-world scenarios where external factors (such as lighting, camera orientation, etc.) can alter the trigger. Most papers consider digital applications where the implanted trigger is nearly unaltered [101; 238].

In conclusion, although some recent work seems to have improved the effectiveness of poisoning attacks, they work on assumptions whose practical evidence is still unclear, limiting their applicability against real-world applications.

8.1.2 Computational Complexity of Poisoning Attacks

The second challenge we discuss here is related to the solution of the bilevel programming problem used to optimize poisoning attacks. The problem, as analyzed by Muñoz-González et al. [197], is that solving the bilevel formulation with a gradient-based approach requires computing and inverting the Hessian matrix associated with the equilibrium conditions of the inner learning problem, which scales cubically in time and quadratically in space with respect to the number of model's parameters. Even if one may exploit rank-one updates to the Hessian matrix, and Hessian-vector products coupled with conjugate descent to speed up the computation of required gradients, the approach remains too computationally demanding to attack modern deep models, where the number of parameters is on the order of millions.

Nevertheless, it is also true that solving the bilevel problem is expected to improve the effectiveness of the attack and its stealthiness against defenses. For example, the bilevel strategy approach is the only one at the state of the art which allows mounting an effective attack in the *training-from-scratch* (TS) setting. Other heuristic approaches, e.g., *feature collision*, are totally ineffective if the feature extractor ϕ is updated during training [101]. For backdoor poisoning, the recent developments in the literature show that bilevel-inspired attacks are more effective and can better counter existing defenses [76; 204; 251]. Moreover, as we can see in Table 3.2, bilevel strategies are the ones for which only a few defenses exist. The reason for this lack is that it is more challenging to identify them as they do not follow a pre-defined behavior. Thus tackling the complexity of the bilevel poisoning problem remains a relevant open challenge to ensure a fairer and scalable evaluation of modern deep models against such attacks.

8.2 Future Development

Building on the open challenges we identified in Sec. 8.1, we formulate some future research challenges that can pave the way toward advancing knowledge in poisoning literature.

8.2.1 Considering Realistic Threat Models

One pertinent challenge arising from the discussion on poisoning attacks in Sec. 8.1 demands considering more realistic threat models and attack scenarios, as also recently pointed out in [241]. While assessing ML models in real-world settings is not straightforward [249], the development of realistic threat models, possibly for individual applications, is still an open issue in ML security, and has so far only received recognition for test-time attacks [102]. We would thus invite the research community to evaluate poisoning attacks under more realistic assumptions, which also take into account the specific application domain.

8.2.2 Designing More Effective and Scalable Attacks

The other challenge we highlighted in Sec. 8.1 is the computational complexity of poisoning attacks when relying on bilevel optimization. However, the same limitation is also encountered in other research domains such as hyperparameter optimization and meta-learning. More concretely, the former is the process of determining the optimal combination of hyperparameters that maximizes the performance of an underlying learning algorithm. On the other hand, meta-learning encompasses feature selection, algorithm selection, learning to learn, or ensemble learning, to which the same reasoning applies. In principle, by imagining poisoning points as an attacker-controlled learning hyperparameters, we could apply the approaches proposed in these two fields to mount an attack. Notably, we find some initial works connecting these two fields with data poisoning. For example, Shen et al. [242] rely in their approach on a k-arms technique, a technique similar to bandits, as done by Jones et al. [136]. Further, Muñoz-González et al. [197] exploited the back-gradient optimization technique proposed in [184; 77], originally proposed for hyperparameter optimization, and subsequently, Huang et al. [129] inherited the same approach making the attack more effective against deep neural networks. Apart from the work just mentioned, the connection between the two fields and poisoning is still currently under-investigated, and other ideas could be explored. For example, the optimization proposed by [178] can further reduce run-time complexity and memory usage even when dealing with millions of hyperparameters. Or another way might be to move away from gradient-based approaches and consider gradient-free approaches, thus overcoming the complexity of the inverting the Hessian matrix seen in Sec. 8.1. In the area of gradient-free methods, the most straightforward way is to use grid or random search [21], which can be sped up using reinforcement learning [159]. Also, Bayesian optimization has been used, given a few sampled points from the objective and constraint functions, to approximate the target function [136]. Last but not least, evolutionary algorithms [305] as well as particle swarm optimization [177] have shown to be successful.

In conclusion, we consider these two domains as possible future research directions to find more effective and scalable poisoning attacks for assessing ML robustness in practice.

8.3 Conclusions

In this thesis, we explored the poisoning threats that compromise the integrity and availability of ML systems. In Chapter 1, we presented our research questions related to different topics such as their categorization, scalability, effectiveness, and influence on practical ML systems. In Chapter 2, we briefly discussed ML and adversarial machine learning, introducing concepts and terminology used in the remaining thesis. In Chapter 3, we systematized data poisoning attacks according to the attacker's thread model and identified possible defenses that may be adopted to counter them. In Chapter 4, we discussed the scalability issues of data poisoning attacks, identifying the necessity of developing more scalable approaches to stage them. To this purpose, we defined BetaPoisoning, a simple but efficient heuristic that can be adapted to craft poisoning samples targeting linear classifiers. We compared our attack against state-of-the-art poisoning attacks highlighting the huge computational gap, favoring our methods. Notably, *BetaPoisoning* reaches the same attack effectiveness of state-of-the-art white-box attacks or exceed them while offering a more scalable solution when dataset dimensionality increase. In Chapter 5, we examined the factors influencing the effectiveness of backdoor poisoning attacks, i.e., their accuracy in predicting the test samples containing backdoor triggers as the attacker desiderata. As a result, we found that regularizing the ML model can improve its robustness against malicious alteration of the training data. However, the attacker could increase their attack strength (e.g., increasing the trigger size or visibility or injecting more poisoning samples), improving the backdoor effectiveness but being less stealthy among inspection. In Chapter 6, we paved the way toward a novel kind of security violation caused by poisoning attacks, i.e., energy-latency attacks. We explained our *sponge poisoning* attack, which tampers with the training algorithm to vanish the effect of hardware acceleration strategies to increase energy consumption and latency at inference time. We further introduced a novel function for measuring energy consumption in ML models, highlighting its improvements compared to the state-of-the-art approaches and suitability in the poisoning context. In Chapter 7, we investigated how an attacker can stage a poisoning attack against the clustering algorithm, assuming the attacker has only oracle access to the target model. We derived a black-box genetic algorithm to optimize the adversarial noise injected into the victim data, and we theoretically and empirically assessed its convergence properties toward global optima. We finally have proposed in this Chapter the open challenging in the poisoning literature and highlighted possible future directions to address them.

Limitations of this Doctoral Dissertation. Our categorization framework, seen in Chapter 3, only considers data poisoning attacks in the computer vision domain. Although it is the domain where the vast majority of work has been done, more domain applications have been proved vulnerable to poisoning, as seen in Sec. 3.5. However, our categorization for attack strategy in that domain requires further thought and extension to encompass them. BetaPoisoning, seen in Chapter 4, is effective only when attacking linear classifiers. Therefore its application remains limited to only the cases where the learning task is linearly separable or when using pre-trained networks that are subsequently fine-tuned on the last layer. Our framework for understating the factors influencing the effectiveness of poisoning attacks, presented in Chapter 5, inherits the same limitations of interpreting the predictions of DNNs with influence functions while working pretty good for convex-learners [144; 18]. Specifically, Basu et al. [18] observed that several factors (i.e., network architecture, hyperparameters, initialization, and loss curvatures) can significantly reduce the quality of influence estimates, thus affecting the integrity of our framework. To overcome this deficiency, we estimated the influence function given by DNNs with finite difference approximation, but further investigation in this direction (e.g., considering more architecture, analyzing the approximation error) is required. Finally, our attack against the clustering algorithm, seen in Chapter 7, has converged, effectively compromising the clustering performance. However, it requires many queries for optimizing the poisoning samples, which may not be affordable for the attacker when the victim dataset scale. The limitations of our works derive from the open challenges we identified for poisoning, and we believe they define interesting future research directions.

Concluding Remarks. The increasing adoption of data-driven models in production systems demands a rigorous analysis of their reliability in the presence of malicious users aiming to compromise them. However, securing learning models is still a long way to go. The open challenges we explored in this Chapter are limiting the development of novel tools for testing and identifying their vulnerabilities before they are deployed in safety-critical applications. This thesis aims to take a step toward this direction by first shedding light on existing types of attacks, categorizing them with respect to their assumptions and attack methodologies. We then investigate four more aspects of poisoning attacks, namely: (i) their scalability issues, (ii) the factors affecting their effectiveness against ML models, (iii) their feasibility against unsupervised clustering algorithms when limiting the attacker's knowledge, and (iv) how poisoning can go beyond misclassification causing energy-latency violations. Finally, we identified the relevant open challenges limiting the advancement of the poisoning literature, i.e. missing scalable and effective attacks and lack of attacks for realistic threat models, together with reasonable research directions that can tackle them. In conclusion, our contribution can help clarify what threats an ML system may encounter when malicious users can influence part of the training pipeline (data gathering or model training). We hope this thesis will foster further research developments in implementing reliable systems even in the presence of data poisoning threats and developing new poisoning benchmarks to test them.

Acknowledgement. The research reported in this PhD thesis has been partially supported by the PRIN 2017 project RexLearn (grant no. 2017TWNMH2), funded by the Italian Ministry of Education, University and Research; and by BMK, BMDW, and the Province of Upper Austria in the frame of the COMET Programme managed by FFG in the COMET Module S3AI.

Bibliography

[1] H. Aghakhani, T. Eisenhofer, L. Schönherr, D. Kolossa, T. Holz, C. Kruegel, and G. Vigna. VENOMAVE: clean-label poisoning against speech recognition. *arXiv:2010.10682*, 2020.

[2] H. Aghakhani, D. Meng, Y.-X. Wang, C. Kruegel, and G. Vigna. Bullseye polytope: A scalable clean-label poisoning attack with improved transferability. In *EuroS&P*, pages 159–178, 2021.

[3] J. Albericio, P. Judd, T. H. Hetherington, T. M. Aamodt, N. D. E. Jerger, and A. Moshovos. Cnvlutin: Ineffectual-neuron-free deep neural network computing. In *ACM/IEEE Annual Int. Symp. on Computer Architecture, ISCA*, 2016.

[4] E. Alpaydin and C. Kaynak. *Optical Recognition of Handwritten Digits*, 1995. URL `https://archive.ics.uci.edu/ml/datasets/optical+recognition+of+handwritten+digits`.

[5] M. Alzantot, Y. Sharma, S. Chakraborty, H. Zhang, C. Hsieh, and M. B. Srivastava. Genattack: practical black-box attacks with gradient-free optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO*, 2019.

[6] D. Arthur and S. Vassilvitskii. k-means++: the advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1027–1035, 2007.

[7] C. Ashcraft and K. Karra. Poisoning deep reinforcement learning agents with in-distribution triggers. *arXiv:2106.07798*, 2021.

[8] Autocomplete Failure. Algorithmic defamation: The case of the shameless autocomplete. http://www.nickdiakopoulos.com/2013/08/06/algorithmic-defamation-the-case-of-the-shameless-autocomplete/, 2013.

[9] M. R. Azghadi, C. Lammie, J. K. Eshraghian, M. Payvand, E. Donati, B. Linares-Barranco, and G. Indiveri. Hardware implementation of deep network accelerators towards healthcare and biomedical applications. *IEEE Trans. Biomed. Circuits Syst.*, 2020.

[10] F. R. Bach, R. Jenatton, J. Mairal, and G. Obozinski. Optimization with sparsity-inducing penalties. *Foundations and Trends in Machine Learning*, 2012.

[11] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov. How to backdoor federated learning. In *The 23rd International Conference on AI and Statistics, AISTATS*, pages 2938–2948. PMLR, 2020.

[12] P. Bajcsy and M. Majurski. Baseline pruning-based approach to trojan detection in neural networks. *arXiv:2101.12016*, 2021.

[13] T. Baluta, S. Shen, S. Shinde, K. S. Meel, and P. Saxena. Quantitative verification of neural networks and its security applications. In *CCS*, 2019.

[14] K. Banihashem, A. Singla, and G. Radanovic. Defense against reward poisoning attacks in reinforcement learning. *arXiv:2102.05776*, 2021.

[15] M. Barni, K. Kallas, and B. Tondi. A new backdoor attack in CNNS by training set corruption without label poisoning. In *IEEE International Conference on Image Proc., ICIP*, pages 101–105. IEEE, 2019.

[16] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar. Can ml be secure? In *2006 ACM Symposium on Inf., Computer and Communications Security, ASIACCS*, pages 16–25. ACM, 2006.

[17] M. Barreno, B. Nelson, A. D. Joseph, and J. D. Tygar. The security of machine learning. *Machine Learning*, 2010.

[18] S. Basu, P. Pope, and S. Feizi. Influence functions in deep learning are fragile. In *ICLR*, 2021.

[19] V. Behzadan and A. Munir. Vulnerability of deep reinforcement learning to policy induction attacks. In *ML and Data Mining in Pattern Recognition - 13th Int. Conf., MLDM 2017*, pages 262–275. Springer, 2017.

[20] A. Bendale and T. E. Boult. Towards open set deep networks. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2016.

[21] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of ML research*, 13(2), 2012.

[22] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. B. Calo. Analyzing federated learning through an adversarial lens. In *International Conference on ML,ICML 2019*. PMLR, 2019.

[23] B. Biggio and F. Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 2018.

[24] B. Biggio and F. Roli. Wild patterns: Ten years after the rise of adversarial ml. *Pattern Recognition*, 84:317–331, 2018.

[25] B. Biggio, I. Corona, G. Fumera, G. Giacinto, and F. Roli. Bagging classifiers for fighting poisoning attacks in adversarial classification tasks. In *Int. workshop on multiple classifier Sys.*, pages 350–359. Springer, 2011.

[26] B. Biggio, B. Nelson, and P. Laskov. Support vector machines under adversarial label noise. In *ACML2011*, pages 97–112, 2011.

[27] B. Biggio, B. Nelson, and P. Laskov. Poisoning attacks against support vector machines. In *ICML 2012*. icml.cc / Omnipress, 2012.

[28] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Srndic, P. Laskov, G. Giacinto, and F. Roli. Evasion attacks against ml at test time. In *ML and Knowl. Disc. in Databases - Eur. Conference,ECML PKDD*. Springer, 2013.

[29] B. Biggio, I. Pillai, S. R. Bulò, D. Ariu, M. Pelillo, and F. Roli. Is data clustering in adversarial settings secure? In *6th ACM Workshop on Art. Intell. and Sec., AISec 2013*, pages 87–98. ACM, 2013.

[30] B. Biggio, S. R. Bulò, I. Pillai, M. Mura, E. Z. Mequanint, M. Pelillo, and F. Roli. Poisoning complete-linkage hierarchical clustering. In *Structural, Syntactic, and Statistical Pattern Recognition, S+SSPR*, 2014.

[31] B. Biggio, K. Rieck, D. Ariu, C. Wressnegger, I. Corona, G. Giacinto, and F. Roli. Poisoning behavioral malware clustering. In *7th ACM Workshop on Art. Intell. and Sec., AISec 2014*, pages 27–36. ACM, 2014.

[32] A. Bojchevski and S. Günnemann. Adversarial attacks on node embeddings via graph poisoning. In *36th International Conference on ML, ICML 2019*, pages 695–704, 2019.

[33] E. Borgnia, V. Cherepanova, L. Fowl, A. Ghiasi, J. Geiping, M. Goldblum, T. Goldstein, and A. Gupta. Strong data augmentation sanitizes poisoning and backdoor attacks without an accuracy tradeoff. In *IEEE ICASSP*, 2021.

[34] E. Borgnia, J. Geiping, V. Cherepanova, L. Fowl, A. Gupta, A. Ghiasi, F. Huang, M. Goldblum, and T. Goldstein. Dp-instahide: Provably defusing poisoning and backdoor attacks with differentially private data augmentations. *arXiv:2103.02079*, 2021.

[35] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Min. Knowl. Discov.*, 2(2):121–167, 1998.

[36] D. Cao, S. Chang, Z. Lin, G. Liu, and D. Sun. Understanding distributed poisoning attack in federated learning. In *25th IEEE International Conference on Parallel and Distributed Sys.*, pages 233–239, 2019.

[37] N. Carlini and D. Wagner. Defensive distillation is not robust to adversarial examples, 2016.

[38] N. Carlini and D. A. Wagner. Towards evaluating the robustness of neural networks. In *Symposium on Security and Privacy, SP*. IEEE Computer Society, 2017.

[39] J. Carnerero-Cano, L. Munoz-González, P. Spencer, and E. C. Lupu. Regularization can help mitigate poisoning attacks... with the right hyperparameters. In *ICLR Workshop on Security and Safety in Machine Learning System*, 2021.

[40] R. Caruana, S. Lawrence, and C. L. Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *NIPS*, 2000.

[41] G. Cauwenberghs and T. Poggio. Incremental and decremental support vector machine learning. *NIPS*, 2001.

[42] A. Chakraborty, M. Alam, V. Dey, A. Chattopadhyay, and D. Mukhopadhyay. Adversarial attacks and defences: A survey. *arXiv:1810.00069*, 2018.

[43] M. Charytanowicz, J. Niewczas, P. Kulczycki, P. A. Kowalski, S. Łukasik, and S. Żak. Complete gradient clustering algorithm for features analysis of x-ray images. In *Information Technologies in Biomedicine*, pages 15–24. Springer Berlin Heidelberg, 2010.

[44] B. Chen, W. Carvalho, N. Baracaldo, H. Ludwig, B. Edwards, T. Lee, I. Molloy, and B. Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. *arXiv:1811.03728*, 2018.

[45] C. Chen and J. Dai. Mitigating backdoor attacks in lstm-based text classification sys. by backdoor keyword identification. *arXiv:2007.12070*, 2020.

[46] H. Chen, C. Fu, J. Zhao, and F. Koushanfar. Deepinspect: A black-box trojan detection and mitigation framework for deep neural networks. In *Int. Joint Conf. on AI, IJCAI 2019*, pages 4658–4664, 2019.

[47] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *10th ACM Workshop on AI and Security*, AISec '17, pages 15–26, 2017.

[48] R. Chen, Z. Li, J. Li, J. Yan, and C. Wu. On collective robustness of bagging against data poisoning. In *ICML*, pages 3299–3319, 2022.

[49] S. Chen, N. Carlini, and D. A. Wagner. Stateful detection of black-box adversarial attacks. *arXiv*, 2019.

[50] X. Chen, C. Liu, B. Li, K. Lu, and D. Song. Targeted backdoor attacks on deep learning sys. using data poisoning. *arXiv:1712.05526*, 2017.

[51] X. Chen, A. Salem, M. Backes, S. Ma, and Y. Zhang. Badnl: Backdoor attacks against nlp models. In *ICML 2021 Workshop on Adversarial ML*, 2021.

[52] Y. Chen, J. S. Emer, and V. Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *ACM/IEEE Annual Int. Symp. on Computer Architecture, ISCA*. IEEE Computer Society, 2016.

[53] H. Cheng, K. Xu, S. Liu, P.-Y. Chen, P. Zhao, and X. Lin. Defending against backdoor attack on deep neural networks. *arXiv:2002.12162*, 2020.

[54] S. Cheng, Y. Liu, S. Ma, and X. Zhang. Deep feature space trojan attack of neural networks by controlled detoxification. In *Thirty-Fifth AAAI Conf. on AI 2021*, pages 1148–1156. AAAI Press, 2021.

[55] A. Chhabra, A. Roy, and P. Mohapatra. Suspicion-free adversarial attacks on clustering algorithms. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI*. AAAI Press, 2020.

[56] Chinese chatbots. Ai getting out of hand? chinese chatbots re-educated after rogue rants. https://www.khaleejtimes.com/technology/ai-getting-out-of-hand-chinese-chatbots-re-educated-after-rogue-rants, 2017.

[57] E. Chou, F. Tramer, and G. Pellegrino. Sentinet: Detecting localized universal attacks against deep learning systems. In *IEEE Security and Privacy Workshops, SPW 2020*, pages 48–54. IEEE, 2020.

[58] A. Christmann and I. Steinwart. On robustness properties of convex risk minimization methods for pattern recognition. *The Journal of Machine Learning Research*, 2004.

[59] A. E. Cinà, K. Grosse, S. Vascon, A. Demontis, B. Biggio, F. Roli, and M. Pelillo. Backdoor learning curves: Explaining backdoor poisoning beyond influence functions. *arXiv:2106.07214*, 2021.

[60] A. E. Cinà, S. Vascon, A. Demontis, B. Biggio, F. Roli, and M. Pelillo. The hammer and the nut: Is bilevel optimization really needed to poison linear classifiers? In *International Joint Conference on Neural Networks, IJCNN 2021*, pages 1–8. IEEE, 2021.

[61] A. E. Cinà, A. Demontis, B. Biggio, F. Roli, and M. Pelillo. Energy-latency attacks via sponge poisoning. *arXiv:2203.08147*, 2022.

[62] A. E. Cinà, K. Grosse, A. Demontis, B. Biggio, F. Roli, and M. Pelillo. Machine learning security against data poisoning: Are we there yet? *CoRR*, abs/2204.05986, 2022.

[63] A. E. Cinà, K. Grosse, A. Demontis, S. Vascon, W. Zellinger, B. A. Moser, A. Oprea, B. Biggio, M. Pelillo, and F. Roli. Wild patterns reloaded: A survey of machine learning security against training data poisoning. *CoRR*, abs/2205.01992, 2022.

[64] A. E. Cinà, A. Torcinovich, and M. Pelillo. A black-box adversarial attack for poisoning clustering. *Pattern Recognition*, 122:108306, 2022.

[65] J. Cohen, E. Rosenfeld, and Z. Kolter. Certified adversarial robustness via randomized smoothing. In *International Conference on Machine Learning, ICML*, pages 1310–1320. PMLR, 2019.

[66] C. Cortes and V. N. Vapnik. Support-vector networks. *Machine Learning*, 1995.

[67] G. Cretu, A. Stavrou, M. Locasto, S. Stolfo, and A. Keromytis. Casting out demons: Sanitizing training data for anomaly sensors. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, 2008.

[68] J. Crussell and W. P. Kegelmeyer. Attacking DBSCAN for fun and profit. In *SIAM International Conference on Data Mining*, 2015.

[69] M. F. Dacrema, P. Cremonesi, and D. Jannach. Are we really making much progress? a worrying analysis of recent neural recommendation approaches. In *ACM*, 2019.

[70] J. de Rooi and P. Eilers. Deconvolution of pulse trains with the l0 penalty. *Analytica Chimica Acta*, 2011.

[71] L. Demetrio, B. Biggio, G. Lagorio, F. Roli, and A. Armando. Functionality-preserving black-box optimization of adversarial windows malware. *IEEE Trans. Inf. Forensics Secur.*, 2021.

[72] A. Demontis, B. Biggio, G. Fumera, G. Giacinto, and F. Roli. Infinity-norm support vector machines against adversarial label contamination. In *ITASEC*, volume 1816 of *CEUR Workshop Proceedings*, pages 106–115. CEUR-WS.org, 2017.

[73] A. Demontis, M. Melis, M. Pintor, M. Jagielski, B. Biggio, A. Oprea, C. Nita-Rotaru, and F. Roli. Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks. In *USENIX Sec. Symp.* USENIX Association, 2019.

[74] I. Diakonikolas, G. Kamath, D. Kane, J. Li, J. Steinhardt, and A. Stewart. Sever: A robust meta-algorithm for stochastic optimization. In *International Conference on ML*, pages 1596–1606. PMLR, 2019.

[75] B. G. Doan, E. Abbasnejad, and D. C. Ranasinghe. Februus: Input purification defense against trojan attacks on deep neural network systems. In *Annual Computer Security Applications Conf.*, pages 897–912, 2020.

[76] K. Doan, Y. Lao, W. Zhao, and P. Li. Lira: Learnable, imperceptible and robust backdoor attacks. In *IEEE/CVF International Conference on Computer Vision, ICCV 2021*, pages 11966–11976, 2021.

[77] J. Domke. Generic methods for optimization-based modeling. In *15th International Conference on Art. Intell. and Statistics, AISTATS 2012*, pages 318–326. JMLR, 2012.

[78] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *ICML*, 2014.

[79] Y. Dong, X. Yang, Z. Deng, T. Pang, Z. Xiao, H. Su, and J. Zhu. Black-box detection of backdoor attacks with limited information and data. In *ICCV*, 2021.

[80] M. Du, R. Jia, and D. Song. Robust anomaly detection and backdoor attack detection via differential privacy. In *International Conference on Learning Representations*, 2020.

[81] J. G. Dutrisac and D. B. Skillicorn. Hiding clusters in adversarial settings. In *IEEE International Conference on Intelligence and Security Informatics, ISI*, 2008.

[82] A. E. Eiben, E. H. L. Aarts, and K. M. van Hee. Global convergence of genetic algorithms: A markov chain analysis. In *Parallel Problem Solving from Nature, 1st Workshop, PPSN*, volume 496 of *Lecture Notes in Computer Science*, pages 4–12. Springer, 1990.

[83] A. E. Eiben, E. H. L. Aarts, and K. M. Van Hee. Global convergence of genetic algorithms: A markov chain analysis. In *Parallel Problem Solving from Nature*. Springer Berlin Heidelberg, 1991.

[84] A. Ess, B. Leibe, K. Schindler, and L. van Gool. Robust multiperson tracking from a mobile platform. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2009.

[85] A. P. et al. Pytorch: An imperative style, high-performance deep learning library. In *Neural Information Processing Systems, NeurIPS*, 2019.

[86] E. S. C. et al. Serving dnns in real time at datacenter scale with project brainwave. *IEEE Micro*, 2018.

[87] K. M. H. et al. Applied machine learning at facebook: A datacenter infrastructure perspective. In *IEEE Int. Symp. on High Performance Computer Architecture, HPCA*, 2018.

[88] N. P. J. et al. In-datacenter performance analysis of a tensor processing unit. In *International Symposium on Computer Architecture, ISCA*. ACM, 2017.

[89] European Commission and Directorate-General for Communications Networks, Content and Technology. *Ethics guidelines for trustworthy AI*. Publications Office, 2019.

[90] T. Everitt, V. Krakovna, L. Orseau, and S. Legg. Reinforcement learning with a corrupted reward channel. In *26th Int. Joint Conf. on AI, IJCAI 2017*, pages 4705–4713, 2017.

[91] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song. Robust physical-world attacks on deep learning visual classification. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2018.

[92] J. Feng, H. Xu, S. Mannor, and S. Yan. Robust logistic regression and classification. In *Advances in Neural Inf. Proc. Sys. 27: Annual Conf. on Neural Inf. Proc. Sys., NIPS*, pages 253–261, 2014.

[93] J. Feng, Q. Cai, and Z. Zhou. Learning to confuse: Generating training time adversarial data with auto-encoder. In *Advances in Neural Inf. Proc. Sys., NeurIPS 2019*, pages 11971–11981, 2019.

[94] L. Fowl, P.-y. Chiang, M. Goldblum, J. Geiping, A. Bansal, W. Czaja, and T. Goldstein. Preventing unauthorized use of proprietary data: Poisoning for secure dataset release. *arXiv:2103.02683*, 2021.

[95] L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, and M. Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In *ICML*, 2018.

[96] C. Frederickson, M. Moore, G. Dawson, and R. Polikar. Attack strength vs. detectability dilemma in adversarial ml. In *International Joint Conference on Neural Networks, IJCNN 2018*, pages 1–8. IEEE, 2018.

[97] Y. Gao, C. Xu, D. Wang, S. Chen, D. C. Ranasinghe, and S. Nepal. Strip: A defence against trojan attacks on deep neural networks. In *35th Annual Computer Security Applications Conf.*, pages 113–125, 2019.

[98] Y. Gao, B. G. Doan, Z. Zhang, S. Ma, J. Zhang, A. Fu, S. Nepal, and H. Kim. Backdoor attacks and countermeasures on deep learning: A comprehensive review. *arXiv:2007.10760*, 2020.

[99] A. Gardner, J. Kanno, C. A. Duncan, and R. R. Selmic. Measuring distance between unordered sets of different sizes. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*. IEEE Computer Society, 2014.

[100] J. Geiping, L. Fowl, G. Somepalli, M. Goldblum, M. Moeller, and T. Goldstein. What doesn't kill you makes you robust (er): Adversarial training against poisons and backdoors. *arXiv:2102.13624*, 2021.

[101] J. Geiping, L. H. Fowl, W. R. Huang, W. Czaja, G. Taylor, M. Moeller, and T. Goldstein. Witches' brew: Industrial scale data poisoning via gradient matching. In *International Conference on Learning Representations, ICLR 2021*. OpenReview, 2021.

[102] J. Gilmer, R. P. Adams, I. Goodfellow, D. Andersen, and G. E. Dahl. Motivating the rules of the game for adversarial example research. *arXiv*, 2018.

[103] D. E. Goldberg. *Genetic Algorithms in Search Optimization and Machine Learning*. Addison-Wesley, 1989. ISBN 0-201-15767-5.

[104] M. Goldblum, D. Tsipras, C. Xie, X. Chen, A. Schwarzschild, D. Song, A. Madry, B. Li, and T. Goldstein. Dataset security for machine learning: Data poisoning, backdoor attacks, and defenses. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2022.

[105] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In *3rd International Conference on Learning Representations, ICLR*, 2015.

[106] Google image search failure. Jewish baby stroller image algorithm. https://www.timebulletin.com/jewish-baby-stroller-image-algorithm/, 2020.

[107] K. Grosse, L. Bieringer, T. R. Besold, B. Biggio, and K. Krombholz. " why do so?"–a practical perspective on machine learning security. *arXiv*, 2022.

[108] T. Gu, B. Dolan-Gavitt, and S. Garg. Badnets: Identifying vulnerabilities in the ml model supply chain. *arXiv:1708.06733*, 2017.

[109] J. Guo and C. Liu. Practical poisoning attacks on neural networks. In *European Conference on Computer Vision, ECCV*, pages 142–158. Springer, 2020.

[110] J. Guo, A. Li, and C. Liu. AEVA: Black-box backdoor detection using adversarial extreme value analysis. In *International Conference on Learning Representations*, 2022.

[111] W. Guo, L. Wang, X. Xing, M. Du, and D. Song. Tabor: A highly accurate approach to inspecting and restoring trojan backdoors in ai systems. *arXiv:1908.01763*, 2019.

[112] N. Y. Hammerla, S. Halloran, and T. Plötz. Deep, convolutional, and recurrent models for human activity recognition using wearables. In *Int. Joint Conf. on Artificial Intelligence, IJCAI*. IJCAI/AAAI Press, 2016.

[113] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. EIE: efficient inference engine on compressed deep neural network. In *ACM/IEEE Int. Symp. on Computer Architecture, ISCA*, 2016.

[114] Y. Han, D. Hubczenko, P. Montague, O. De Vel, T. Abraham, B. I. Rubinstein, C. Leckie, T. Alpcan, and S. Erfani. Adversarial reinforcement learning under partial observability in autonomous computer network defence. In *Int. Joint Conf. on Neural Networks, IJCNN 2020*, pages 1–8. IEEE, 2020.

[115] J. A. Hartigan and M. A. Wong. A k-means clustering algorithm. *Journal of the Royal Statistical Society*, 1979.

[116] A. B. A. Hassanat, K. Almohammadi, E. Alkafaween, E. Abunawas, A. M. Hammouri, and V. B. S. Prasath. Choosing mutation and crossover ratios for genetic algorithms - A review with a new dynamic approach. *Inf.*, 10(12), 2019.

[117] T. Hastie, S. Rosset, R. Tibshirani, and J. Zhu. The entire regularization path for the support vector machine. *JMLR*, 2004.

[118] J. Hayase, W. Kong, R. Somani, and S. Oh. Spectre: Defending against backdoor attacks using robust statistics. In *International Conference on ML, ICML 2020*, pages 4129–4139. PMLR, 2021.

[119] J. Hayes and O. Ohrimenko. Contamination attacks and mitigation in multi-party ml. *Advances in Neural Inf. Proc. Sys., NIPS 2018*, 31:6604–6615, 2018.

[120] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European Conf. on Computer Vision, ECCV*, Lecture Notes in Computer Science. Springer, 2016.

[121] A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 2000.

[122] S. Hong, V. Chandrasekaran, Y. Kaya, T. Dumitraş, and N. Papernot. On the effectiveness of mitigating data poisoning attacks with gradient shaping. *arXiv:2002.11497*, 2020.

[123] R. A. Horn and C. R. Johnson. *Matrix Analysis.* Cambridge University Press, USA, 2nd edition, 2012.

[124] D. W. Hosmer and S. Lemeshow. Applied logistic regression. *Wiley Series in Probability and Statistics*, 1989.

[125] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In *Int. Joint Conf. on Neural Networks*, 2013.

[126] J. Howard. Imagenette. https://github.com/fastai/imagenette/, 2020.

[127] X. Hu, X. Lin, M. Cogswell, Y. Yao, S. Jha, and C. Chen. Trigger hunting with a topological prior for trojan detection. In *International Conference on Learning Representations*, 2022.

[128] K. Huang, Y. Li, B. Wu, Z. Qin, and K. Ren. Backdoor defense via decoupling the training process. In *International Conference on Learning Representations*, 2022.

[129] W. R. Huang, J. Geiping, L. Fowl, G. Taylor, and T. Goldstein. Metapoison: Practical general-purpose clean-label data poisoning. In *Advances in Neural Inf. Proc. Sys., NeurIPS 2020*, 2020.

[130] X. Huang, M. Alzantot, and M. Srivastava. Neuroninspect: Detecting backdoors in neural networks via output explanations. *arXiv:1911.07399*, 2019.

[131] Y. Huang and Q. Zhu. Deceptive reinforcement learning under adversarial manipulations on cost signals. In *International Conference on Decision and Game Theory for Security*, pages 217–237. Springer, 2019.

[132] L. Hubert and P. Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.

[133] M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li. Manipulating ml: Poisoning attacks and countermeasures for regression learning. In *IEEE Symposium on Security and Privacy, S&P*. IEEE, 2018.

[134] M. Jagielski, G. Severi, N. Pousette Harger, and A. Oprea. Subpopulation data poisoning attacks. In *ACM SIGSAC Conference on Computer and Communications Security, CCS 2021*, pages 3104–3122, 2021.

[135] J. Jia, X. Cao, and N. Z. Gong. Certified robustness of nearest neighbors against data poisoning attacks. *arXiv:2012.03765*, 2020.

[136] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 1998.

[137] P. Judd, A. D. Lascorz, S. Sharify, and A. Moshovos. Cnvlutin2: Ineffectual-activation-and-weight-free deep neural network computing. *CoRR*, 2017.

[138] M. Juuti, S. Szyller, A. Dmitrenko, S. Marchal, and N. Asokan. Prada: Protecting against dnn model stealing attacks. *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 512–527, 2019.

[139] S. Kaviani and I. Sohn. Defense against neural trojan attacks: A survey. *Neurocomputing*, 423:651–667, 2021.

[140] D. Kim, J. Ahn, and S. Yoo. A novel zero weight/activation-aware hardware architecture of convolutional neural network. In *Design, Automation & Test in Europe Conf. & Exhibition.* IEEE, 2017.

[141] P. Kiourti, K. Wardega, S. Jha, and W. Li. Trojdrl: evaluation of backdoor attacks on deep reinforcement learning. In *ACM/IEEE Design Automation Conf., DAC 2020*, pages 1–6. IEEE, 2020.

[142] M. Kloft and P. Laskov. Online anomaly detection under adversarial impact. In *AISTATS 2010*, pages 405–412. JMLR, 2010.

[143] S. Koffas, J. Xu, M. Conti, and S. Picek. Can you hear it? backdoor attacks via ultrasonic triggers. *arXiv:2107.14569*, 2021.

[144] P. W. Koh and P. Liang. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning, ICML 2017*, pages 1885–1894. PMLR, 2017.

[145] S. Kolouri, A. Saha, H. Pirsiavash, and H. Hoffmann. Universal litmus patterns: Revealing backdoor attacks in cnns. In *IEEE/CVF International Conference on Computer Vision, ICCV 2021*, pages 301–310, 2020.

[146] Korean ChatBot Failure. Ai chatbot shut down after learning to talk like a racist asshole. https://www.vice.com/en/article/akd4g5/ai-chatbot-shut-down-after-learning-to-talk-like-a-racist-asshole, 2021.

[147] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[148] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 2012.

[149] J. Kukacka, V. Golkov, and D. Cremers. Regularization for deep learning: A taxonomy. *CoRR*, 2017.

[150] R. S. S. Kumar, D. R. O'Brien, K. Albert, S. Viljöen, and J. Snover. Failure modes in machine learning systems. *CoRR*, abs/1911.11034, 2019.

[151] R. S. S. Kumar, M. Nyström, J. Lambert, A. Marshall, M. Goertzel, A. Comissoneru, M. Swann, and S. Xia. Adversarial machine learning-industry perspectives. In *IEEE Security and Privacy Workshops, SPW 2020*. IEEE, 2020.

[152] A. Kurakin, I. J. Goodfellow, and S. Bengio. Adversarial machine learning at scale. In *ICLR*, 2017.

[153] R. Laishram and V. V. Phoha. Curie: A method for protecting svm classifier from poisoning attack. *arXiv:1606.01584*, 2016.

[154] T. K. Landauer, P. W. Foltz, and D. Laham. An introduction to latent semantic analysis. *Discourse Processes*, 25(2-3):259–284, 1998.

[155] K. Lang. Newsweeder: Learning to filter netnews. In *Machine Learning, Proceedings of the Twelfth International Conference on Machine Learning, ICML*, 1995.

[156] Learning from Tay. Learning from tay's introduction - the official microsoft blog. https://blogs.microsoft.com/blog/2016/03/25/learning-tays-introduction/, 2016.

[157] Y. LeCun, C. Cortes, and C. Burges. Mnist handwritten digit database. *ATT Labs Available: http://yann.lecun.com/exdb/mnist*, 2010.

[158] A. Levine and S. Feizi. Deep partition aggregation: Provable defenses against general poisoning attacks. In *International Conference on Learning Representations, ICLR 2021*, 2021.

[159] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of ML Research*, 18 (1):6765–6816, 2017.

[160] S. Li, M. Xue, B. Zhao, H. Zhu, and X. Zhang. Invisible backdoor attacks on deep neural networks via steganography and regularization. *IEEE Trans. on Dependable and Secure Computing*, pages 1–1, 2020.

[161] X. Li and F. Li. Adversarial examples detection in deep networks with convolutional filter statistics. In *IEEE International Conference on Computer Vision, ICCV*, 2017.

[162] Y. Li, B. Wu, Y. Jiang, Z. Li, and S.-T. Xia. Backdoor learning: A survey. *arXiv:2007.08745*, 2020.

[163] Y. Li, T. Zhai, B. Wu, Y. Jiang, Z. Li, and S. Xia. Rethinking the trigger of backdoor attack. *arXiv:2004.04692*, 2020.

[164] Y. Li, N. Koren, L. Lyu, X. Lyu, B. Li, and X. Ma. Neural attention distillation: Erasing backdoor triggers from deep neural networks. In *ICLR*, 2021.

[165] Y. Li, Y. Li, B. Wu, L. Li, R. He, and S. Lyu. Invisible backdoor attack with sample-specific triggers. In *IEEE/CVF International Conference on Computer Vision, ICCV 2021*, pages 16463–16472, 2021.

[166] Y. Li, X. Lyu, N. Koren, L. Lyu, B. Li, and X. Ma. Anti-backdoor learning: Training clean models on poisoned data. In *NeurIPS*, volume 34, 2021.

[167] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 2021.

[168] C. Liu, B. Li, Y. Vorobeychik, and A. Oprea. Robust linear regression against training data poisoning. In *10th ACM Workshop on Art. Intell. and Sec., AISec 2017*, pages 91–102. ACM, 2017.

[169] K. Liu, B. Dolan-Gavitt, and S. Garg. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *Int. Symposium on Research in Attacks, Intrusions, and Defenses*, pages 273–294. Springer, 2018.

[170] X. Liu, S. Si, J. Zhu, Y. Li, and C. Hsieh. A unified framework for data poisoning attack to graph-based semi-supervised learning. In *Advances in Neural Inf. Proc. Sys., NeurIPS 2019*, pages 9777–9787, 2019.

[171] Y. Liu, X. Chen, C. Liu, and D. Song. Delving into transferable adversarial examples and black-box attacks. In *5th International Conference on Learning Representations, ICLR*, 2017.

[172] Y. Liu, Y. Xie, and A. Srivastava. Neural trojans. In *IEEE International Conference on Computer Design, ICCD 2017*, pages 45–48, 2017.

[173] Y. Liu, S. Ma, Y. Aafer, W. Lee, J. Zhai, W. Wang, and X. Zhang. Trojaning attack on neural networks. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018*, pages 45–48, 2018.

[174] Y. Liu, W.-C. Lee, G. Tao, S. Ma, Y. Aafer, and X. Zhang. Abs: Scanning neural networks for back-doors by artificial brain stimulation. In *ACM SIGSAC Conf. on Computer and Communications Security, CCS 2019*, pages 1265–1282. ACM, 2019.

[175] Y. Liu, X. Ma, J. Bailey, and F. Lu. Reflection backdoor: A natural backdoor attack on deep neural networks. In *European Conference on Computer Vision - ECCV*, pages 182–199. Springer, 2020.

[176] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *IEEE International Conference on Computer Vision, ICCV*. IEEE Computer Society, 2015.

[177] P. R. Lorenzo, J. Nalepa, M. Kawulok, L. S. Ramos, and J. R. Pastor. Particle swarm optimization for hyper-parameter selection in deep neural networks. In *the genetic and evolutionary computation conference*, pages 481–488, 2017.

[178] J. Lorraine, P. Vicol, and D. Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. In *International Conference on AI and Statistics*, pages 1540–1552. PMLR, 2020.

[179] D. Lowd and C. Meek. Adversarial learning. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2005.

[180] D. Lowd and C. Meek. Good word attacks on statistical spam filters. In *CEAS 2005 - Second Conference on Email and Anti-Spam*, 2005.

[181] Y. Ma, K.-S. Jun, L. Li, and X. Zhu. Data poisoning attacks in contextual bandits. In *International Conference on Decision and Game Theory for Security*, pages 186–204. Springer, 2018.

[182] Y. Ma, X. Zhu, and J. Hsu. Data poisoning against differentially-private learners: Attacks and defenses. In *28th International Joint Conference on AI, IJCAI 2019*, pages 4732–4738, 2019.

[183] R. Machupalli, M. Hossain, and M. Mandal. Review of asic accelerators for deep neural network. *Microprocessors and Microsystems*, 2022.

[184] D. Maclaurin, D. Duvenaud, and R. P. Adams. Gradient-based hyperparameter optimization through reversible learning. In *32nd International Conference on ML, ICML 2015*, pages 2113–2122. JMLR, 2015.

[185] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations, ICLR*. OpenReview.net, 2018.

[186] B. Mallikarjunappa and D. Prabhakar. A novel method of spam mail detection using text based clustering approach. *International Journal of Computer Applications*, 2010.

[187] S. Mambou, P. Maresova, O. Krejcar, A. Selamat, and K. Kuca1. Breast cancer detection using infrared thermal imaging and a deep learning model. *Sensors*, 18, 2018.

[188] W. S. McCulloch and W. H. Pitts. A logical calculus of the ideas immanent in nervous activity. In *The Philosophy of Artificial Intelligence*. Oxford University Press, 1990.

[189] S. McGregor. Preventing repeated real world ai failures by cataloging incidents: The ai incident database. *arXiv:2011.08512*, 2020.

[190] P. Mehta, M. Bukov, C.-H. Wang, A. G. Day, C. Richardson, C. K. Fisher, and D. Schwab. A high-bias, low-variance introduction to machine learning for physicists. *Physics reports*, 2019.

[191] S. Mei and X. Zhu. Using machine teaching to identify optimal training-set attacks on machine learners. In *AAAI*, pages 2871–2877. AAAI Press, 2015.

[192] M. Meila. Comparing clusterings: an axiomatic view. In *Machine Learning, Proceedings of the Twenty-Second International Conference, ICML*, pages 577–584, 2005.

[193] T. A. Meyer and B. Whateley. Spambayes: Effective open-source, bayesian based, email classification system. In *CEAS*, 2004.

[194] A. Moghar and M. Hamiche. Stock market prediction using LSTM recurrent neural network. In *International Conference on Ambient Systems, Networks and Technologies ANT*, volume 170. Elsevier, 2020.

[195] S. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard. Universal adversarial perturbations. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2017.

[196] N. M. Müller, D. Kowatsch, and K. Böttinger. Data poisoning attacks on regression learning and corresponding defenses. In *25th IEEE Pacific Rim Int. Symposium on Dependable Computing, PRDC 2020*, pages 80–89. IEEE, 2020.

[197] L. Muñoz-González, B. Biggio, A. Demontis, A. Paudice, V. Wongrassamee, E. C. Lupu, and F. Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. In *10th ACM Workshop on Art. Intell. and Sec., AISec 2017*, pages 27–38. ACM, 2017.

[198] B. K. Natarajan. Sparse approximate solutions to linear systems. *SIAM J. Comput.*, 1995.

[199] J. F. Navarro, C. S. Frenk, and S. D. M. White. A universal density profile from hierarchical clustering. *The Astrophysical Journal*, 1997.

[200] B. Nelson, M. Barreno, F. J. Chi, A. D. Joseph, B. I. P. Rubinstein, U. Saini, C. Sutton, J. D. Tygar, and K. Xia. Exploiting ml to subvert your spam filter. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats, LEET 2008*, pages 1–9. USENIX, 2008.

[201] B. Nelson, M. Barreno, F. J. Chi, A. D. Joseph, B. I. Rubinstein, U. Saini, C. Sutton, J. Tygar, and K. Xia. Misleading learners: Co-opting your spam filter. In *ML in Cyber Trust*, pages 17–51. Springer, 2009.

[202] J. Newsome, B. Karp, and D. X. Song. Paragraph: Thwarting signature learning by training maliciously. In *RAID 2006*, volume 4219 of *Lecture Notes in Computer Science*, pages 81–105. Springer, 2006.

[203] N. Nguyen and R. Caruana. Consensus clusterings. In *Proceedings of the 7th IEEE International Conference on Data Mining ICDM*, pages 607–612. IEEE Computer Society, 2007.

[204] T. A. Nguyen and A. Tran. Input-aware dynamic backdoor attack. In *Advances in Neural Inf. Proc. Sys., NeurIPS 2020*, 2020.

[205] T. A. Nguyen and A. T. Tran. Wanet - imperceptible warping-based backdoor attack. In *International Conference on Learning Representations, ICLR*. OpenReview.net, 2021.

[206] X. V. Nguyen, J. Epps, and J. Bailey. Information theoretic measures for clusterings comparison: is a correction for chance necessary? In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML*, pages 1073–1080, 2009.

[207] E. Nurvitadhi, D. Sheffield, J. Sim, A. K. Mishra, G. Venkatesh, and D. Marr. Accelerating binarized neural networks: Comparison of fpga, cpu, gpu, and ASIC. In *International Conference on Field-Programmable Technology*. IEEE, 2016.

[208] D. W. Opitz and R. Maclin. Popular ensemble methods: An empirical study. *Journal Artificial Intelligence Research*, 11, 1999.

[209] M. R. Osborne, B. Presnell, and B. A. Turlach. On the lasso and its dual. *J. of Computational and Graphical Statistics*, 2000.

[210] N. Papernot, P. McDaniel, and I. Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv:1605.07277*, 2016.

[211] N. Papernot, P. D. McDaniel, I. J. Goodfellow, S. Jha, Z. B. Celik, and A. Swami. Practical black-box attacks against ml. In *ACM Asia Conference on Computer and Communications Security, AsiaCCS 2017*, pages 506–519. ACM, 2017.

[212] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. S. Emer, S. W. Keckler, and W. J. Dally. SCNN: an accelerator for compressed-sparse convolutional neural networks. In *Int. Symp. on Computer Architecture, ISCA*. ACM, 2017.

[213] G. Pasquale, C. Ciliberto, F. Odone, L. Rosasco, and L. Natale. Teaching icub to recognize objects using deep convolutional neural networks. In *MLIS@ICML*, JMLR Workshop and Conf. Proceedings, 2015.

[214] A. Paudice, L. Muñoz-González, and E. C. Lupu. Label sanitization against label flipping poisoning attacks. In *Joint Eur. Conf. on ML and Knowledge Discovery in Databases*, pages 5–15. Springer, 2018.

[215] F. Pedregosa. Hyperparameter optimization with approximate gradient. In *ICML*. PMLR, 2016.

[216] M. Pelillo and T. Scantamburlo, editors. *Machines We Trust: Perspectives on Dependable AI*. The MIT Press, 2021.

[217] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro. Tesseract: Eliminating experimental bias in malware classification across space and time. In *USENIX Security Symposium*, 2019.

[218] R. Perdisci, D. Dagon, W. Lee, P. Fogla, and M. Sharif. Misleading worm signature generators using deliberate noise injection. In *IEEE Symposium on Security and Privacy, S& P 2006*, pages 15 pp.–31, 2006.

[219] N. Peri, N. Gupta, W. R. Huang, L. Fowl, C. Zhu, S. Feizi, T. Goldstein, and J. P. Dickerson. Deep k-nn defense against clean-label data poisoning attacks. In *Eur. Conf. on Computer Vision*, pages 55–70. Springer, 2020.

[220] F. Qi, Y. Chen, M. Li, Z. Liu, and M. Sun. Onion: A simple and effective defense against textual backdoor attacks. *arXiv:2011.10369*, 2020.

[221] N. Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 1999.

[222] X. Qiao, Y. Yang, and H. Li. Defending neural backdoors via generative distribution modeling. In *Advances in Neural Inf. Proc. Sys., NeurIPS 2019*, pages 14004–14013, 2019.

[223] A. Rakhsha, G. Radanovic, R. Devidze, X. Zhu, and A. Singla. Policy teaching via environment poisoning: Training-time adversarial attacks against reinforcement learning. In *International Conference on ML, ICML 2020*, pages 7974–7984. PMLR, 2020.

[224] S. Romano, X. V. Nguyen, J. Bailey, and K. Verspoor. Adjusting for chance clustering comparison measures. *Journal Machine Learning Research*, 17:134:1–134:32, 2016.

[225] E. Rosenfeld, E. Winston, P. Ravikumar, and Z. Kolter. Certified robustness to label-flipping attacks via randomized smoothing. In *International Conference on ML*, pages 8230–8241. PMLR, 2020.

[226] P. J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20, 1987.

[227] B. I. Rubinstein, B. Nelson, L. Huang, A. D. Joseph, S.-h. Lau, S. Rao, N. Taft, and J. D. Tygar. Antidote: understanding and defending against poisoning of anomaly detectors. In *9th ACM SIGCOMM Conf. on Internet Measurement*, pages 1–14, 2009.

[228] G. Rudolph. *Convergence properties of evolutionary algorithms.* Kovac, 1997.

[229] A. Saha, A. Subramanya, and H. Pirsiavash. Hidden trigger backdoor attacks. In *The Thirty-Fourth AAAI Conf. on AI, AAAI 2020*, pages 11957–11965. AAAI Press, 2020.

[230] A. Salem, R. Wen, M. Backes, S. Ma, and Y. Zhang. Dynamic backdoor attacks against ml models. *arXiv:2003.03675*, 2020.

[231] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manag.*, 24(5):513–523, 1988.

[232] P. Samangouei, M. Kabkab, and R. Chellappa. Defense-gan: Protecting classifiers against adversarial attacks using generative models. In *International Conference on Learning Representations, ICLR*, 2018.

[233] E. Sarkar, Y. Alkindi, and M. Maniatakos. Backdoor suppression in neural networks using input fuzzing and majority voting. *IEEE Design & Test*, 37:103–110, 2020.

[234] E. Sarkar, H. Benkraouda, and M. Maniatakos. Facehack: Triggering backdoored facial recognition sys. using facial characteristics. *arXive:2006.11623*, 2020.

[235] L. Schott, J. Rauber, M. Bethge, and W. Brendel. Towards the first adversarially robust neural network model on mnist. In *International Conference on Learning Representations*, 2018.

[236] A. Schwarzschild, M. Goldblum, A. Gupta, J. P. Dickerson, and T. Goldstein. Just how toxic is data poisoning? A unified benchmark for backdoor and data poisoning attacks. In M. Meila and T. Zhang, editors, *International Conference on Machine Learning, ICML*. PMLR, 2021.

[237] G. Severi, J. Meyer, S. Coull, and A. Oprea. Explanation-guided backdoor poisoning attacks against malware classifiers. In *30th USENIX Security Symposium, USENIX Security 2021*, 2021.

[238] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. In *Advances in Neural Inf. Proc. Sys., NeurIPS 2018*, pages 6106–6116, 2018.

[239] S. Shan, A. N. Bhagoji, H. Zheng, and B. Y. Zhao. Traceback of targeted data poisoning attacks in neural networks. In *USENIX Symposium.* USENIX Association, 2022.

[240] M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *ACM SIGSAC Conference on Computer and Communications Security, CCS*, pages 1528–1540. ACM, 2016.

[241] V. Shejwalkar, A. Houmansadr, P. Kairouz, and D. Ramage. Back to the drawing board: A critical evaluation of poisoning attacks on federated learning. In *IEEE Symposium on Security and Privacy*, 2022.

[242] G. Shen, Y. Liu, G. Tao, S. An, Q. Xu, S. Cheng, S. Ma, and X. Zhang. Backdoor scanning for deep neural networks through k-arm optimization. *arXiv:2102.05123*, 2021.

[243] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8), 2000.

[244] C. Shorten and T. M. Khoshgoftaar. A survey on image data augmentation for deep learning. *J. Big Data*, 2019.

[245] I. Shumailov, Y. Zhao, D. Bates, N. Papernot, R. D. Mullins, and R. Anderson. Sponge examples: Energy-latency attacks on neural networks. In *IEEE European Symposium on Security and Privacy, EuroS&P*, pages 212–231. IEEE, 2021.

[246] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations, ICLR*, 2015.

[247] D. B. Skillicorn. Adversarial knowledge discovery. *IEEE Intelligent Systems*, 2009.

[248] D. Solans, B. Biggio, and C. Castillo. Poisoning attacks on algorithmic fairness. In *ML and Knowledge Discovery in Databases - Eur. Conference, ECML PKDD 2020*, pages 162–177. Springer, 2020.

[249] R. Sommer and V. Paxson. Outside the closed world: On using ml for network intrusion detection. In *IEEE S&P 2010*, pages 305–316. IEEE, 2010.

[250] E. Soremekun, S. Udeshi, S. Chattopadhyay, and A. Zeller. Exposing backdoors in robust ml models. *arXiv:2003.00865*, 2020.

[251] H. Souri, M. Goldblum, L. Fowl, R. Chellappa, and T. Goldstein. Sleeper agent: Scalable hidden trigger backdoors for neural networks trained from scratch. *arXiv:2106.08970*, 2021.

[252] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal Machine Learning Research*, 2014.

[253] J. Steinhardt, P. W. Koh, and P. Liang. Certified defenses for data poisoning attacks. In *Advances in Neural Inf. Proc. Sys. 30: Annual Conf. on Neural Inf. Proc. Sys., NIPS 2017*, pages 3517–3529, 2017.

[254] A. Strehl and J. Ghosh. Cluster ensembles – a knowledge reuse framework for combining multiple partitions. *Journal on Machine Learning Research (JMLR)*, 3, 2002.

[255] J. Su, D. V. Vargas, and K. Sakurai. One pixel attack for fooling deep neural networks. *IEEE Trans. Evol. Comput.*, 2019.

[256] O. Suciu, R. Marginean, Y. Kaya, H. Daume III, and T. Dumitras. When does ml {FAIL}? generalized transferability for evasion and poisoning attacks. In *27th {USENIX} Security Symposium {USENIX} Security 1208*, pages 1299–1316, 2018.

[257] L. Sun, Y. Dou, C. Yang, J. Wang, P. S. Yu, L. He, and B. Li. Adversarial attack and defense on graph data: A survey. *arXiv:1812.10528*, 2018.

[258] M. Sun, Z. Li, C. Xiao, H. Qiu, B. Kailkhura, M. Liu, and B. Li. Can shape structure features improve model robustness under diverse adversarial settings? In *IEEE CVF International Conference on Computer Vision*, pages 7526–7535, 2021.

[259] Z. Sun, P. Kairouz, A. T. Suresh, and H. B. McMahan. Can you really backdoor federated learning? *arXiv:1911.07963*, 2019.

[260] F. Suya, S. Mahloujifar, A. Suri, D. Evans, and Y. Tian. Model-targeted poisoning attacks with provable convergence. In *ICML*. PMLR, 2021.

[261] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *2nd International Conference on Learning Representations, ICLR*, 2014.

[262] D. Tabernik and D. Skocaj. Deep learning for large-scale traffic-sign detection and recognition. *IEEE Transactions on Intelligent Transportation Systems*, 21:1427–1440, 2020.

[263] R. Taheri, R. Javidan, M. Shojafar, Z. Pooranian, A. Miri, and M. Conti. On defending against label flipping attacks on malware detection system. *Neural Computing and Applications*, pages 1–20, 2020.

[264] D. Tang, X. Wang, H. Tang, and K. Zhang. Demon in the variant: Statistical analysis of {DNNs} for robust backdoor contamination detection. In *USENIX Security Symposium (USENIX Security 21)*, pages 1541–1558, 2021.

[265] M. Tapaswi, M. T. Law, and S. Fidler. Video face clustering with unknown number of clusters. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV*, 2019.

[266] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu. Data poisoning attacks against federated learning sytems. In *Eur. Symposium on Research in Computer Security, ESORICS*, pages 480–501. Springer, 2020.

[267] F. Tramèr and D. Boneh. Differentially private learning needs better features (or much more data). In *ICLR*, 2021.

[268] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Stealing ml models via prediction apis. In *USENIX Security Symposium 2016*, pages 601–618, Austin, TX, 2016.

[269] B. Tran, J. Li, and A. Madry. Spectral signatures in backdoor attacks. In *Conference on Neural Information Processing Systems, NIPS 2018*, 2018.

[270] A. Turner, D. Tsipras, and A. Madry. Label-consistent backdoor attacks. *arXiv:1912.02771*, 2019.

[271] S. Udeshi, S. Peng, G. Woo, L. Loh, L. Rawshan, and S. Chattopadhyay. Model agnostic defence against backdoor attacks in ml. *arXiv:1908.02203*, 2019.

[272] A. K. Veldanda, K. Liu, B. Tan, P. Krishnamurthy, F. Khorrami, R. Karri, B. Dolan-Gavitt, and S. Garg. Nnoculation: Catching badnets in the wild. In *14th ACM Workshop on AI and Security*, pages 49–60, 2021.

[273] M. Villarreal-Vasquez and B. Bhargava. Confoc: Content-focus protection against trojan attacks on neural networks. *arXiv:2007.00711*, 2020.

[274] U. von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[275] U. von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 2007.

[276] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *IEEE Symposium on Security and Privacy, S&P 2019*, pages 707–723. IEEE, 2019.

[277] J. Wang, Y. Liu, and B. Li. Reinforcement learning with perturbed rewards. In *AAAI Conf. on Art. Intell.*, pages 6202–6209. AAAI Press, 2020.

[278] W. Wang, A. J. Levine, and S. Feizi. Improved certified defenses against data poisoning with (Deterministic) finite aggregation. In *ICML*, pages 22769–22783. PMLR, 2022.

[279] J. H. Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301), 1963.

[280] M. Weber, X. Xu, B. Karlas, C. Zhang, and B. Li. Rab: Provable robustness against backdoor attacks. *arXiv:2003.08904*, 2020.

[281] J. Wen, B. Z. H. Zhao, M. Xue, A. Oprea, and H. Qian. With great dispersion comes greater resilience: Efficient poisoning attacks and defenses for linear regression models. *IEEE Trans. on Inf. Forensics and Security*, 2021.

[282] E. Wenger, J. Passananti, A. N. Bhagoji, Y. Yao, H. Zheng, and B. Y. Zhao. Backdoor attacks against deep learning sys. in the physical world. *IEEE/CVF International Conference on Computer Vision, ICCV 2021*, pages 6202–6211, 2021.

[283] J. Weston, A. Elisseeff, B. Schölkopf, and M. E. Tipping. Use of the zero-norm with linear models and kernel methods. *Journal Machine Learnearning Research*, 2003.

[284] D. Wu and Y. Wang. Adversarial neuron pruning purifies backdoored deep models. In *NeurIPS 2021*, 2021.

[285] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. F. M. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. S. Steinbach, D. J. Hand, and D. Steinberg. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14:1–37, 2007.

[286] Z. Xiang, D. J. Miller, and G. Kesidis. A benchmark study of backdoor data poisoning defenses for deep neural network classifiers and a novel defense. In *IEEE 29th Int. Workshop on ML for Signal Proc., MLSP 2019*, pages 1–6. IEEE, 2019.

[287] Z. Xiang, D. J. Miller, and G. Kesidis. Detection of backdoors in trained classifiers without access to the training set. *IEEE Trans. on Neural Networks and Learning Sys.*, 2020.

[288] Z. Xiang, D. J. Miller, and G. Kesidis. L-red: Efficient post-training detection of imperceptible backdoor attacks without access to the training set. In *IEEE International Conference on Acoustics, Speech and Signal Proc., ICASSP 2021*, pages 3745–3749. IEEE, 2021.

[289] Z. Xiang, D. J. Miller, and G. Kesidis. Reverse engineering imperceptible backdoor attacks on deep neural networks for detection and training set cleansing. *Computers & Security*, 106: 102280, 2021.

[290] Z. Xiang, D. Miller, and G. Kesidis. Post-training detection of backdoor attacks for two-class and multi-attack scenarios. In *International Conference on Learning Representations*, 2022.

[291] C. Xiao, X. Pan, W. He, J. Peng, M. Sun, J. Yi, M. Liu, B. Li, and D. Song. Characterizing attacks on deep reinforcement learning. *arXiv:1907.09470*, 2019.

[292] H. Xiao, H. Xiao, and C. Eckert. Adversarial label flips attack on support vector machines. In *Eur. Conference on AI. Including Prestigious Applications of AI, ECAI*, pages 870–875. IOS Press, 2012.

[293] H. Xiao, B. Biggio, G. Brown, G. Fumera, C. Eckert, and F. Roli. Is feature selection secure against training data poisoning? In *32nd International Conference on ML, ICML 2015*, pages 1689–1698. JMLR, 2015.

[294] H. Xiao, B. Biggio, B. Nelson, H. Xiao, C. Eckert, and F. Roli. Support vector machines under adversarial label contamination. *Neurocomputing*, 160:53–62, 2015.

[295] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.

[296] C. Xie, K. Huang, P. Chen, and B. Li. DBA: distributed backdoor attacks against federated learning. In *International Conference on Learning Representations, ICLR 2020*. OpenReview, 2020.

[297] A. Xu, Z. Liu, Y. Guo, V. Sinha, and R. Akkiraju. A new chatbot for customer service on social media. *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 2017.

[298] H. Xu, R. Wang, L. Raizman, and Z. Rabinovich. Transferable environment poisoning: Training-time attack on reinforcement learning. In *20th International Conference on Autonomous Agents and MultiAgent Sys.*, pages 1398–1406, 2021.

[299] J. Xu, Z. Li, B. Du, M. Zhang, and J. Liu. Reluplex made more practical: Leaky relu. *IEEE Symp. on Computers and Communications (ISCC)*, 2020.

[300] X. Xu, Q. Wang, H. Li, N. Borisov, C. A. Gunter, and B. Li. Detecting AI trojans using meta neural analysis. In *IEEE Symposium on Security and Privacy, S&P 2021*, pages 103–120. IEEE, 2021.

[301] C. Yang, Q. Wu, H. Li, and Y. Chen. Generative poisoning attack method against neural networks. *CoRR*, abs/1703.01340, 2017.

[302] Y. Yang, T. Y. Liu, and B. Mirzasoleiman. Not all poisons are created equal: Robust training against data poisoning. In *ICML*, pages 25154–25165. PMLR, 2022.

[303] Y. Yao, H. Li, H. Zheng, and B. Y. Zhao. Latent backdoor attacks on deep neural networks. In *ACM SIGSAC Conf. on Computer and Communications Security, CCS 2019*, pages 2041–2055, 2019.

[304] K. Yoshida and T. Fujino. Disabling backdoor and identifying poison data by using knowledge distillation in backdoor attacks on deep neural networks. In *13th ACM Workshop on AI and Security*, pages 117–127, 2020.

[305] S. R. Young, D. C. Rose, T. P. Karnowski, S.-H. Lim, and R. M. Patton. Optimizing deep learning hyper-parameters through an evolutionary algorithm. In *Workshop on ML in High-Performance Computing Environments*, pages 1–5, 2015.

[306] L. Zelnik-Manor and P. Perona. Self-tuning spectral clustering. In *Advances in Neural Information Processing Systems 17 [Neural Information Processing Systems, NIPS*, pages 1601–1608, 2004.

[307] Y. Zeng, H. Qiu, S. Guo, T. Zhang, M. Qiu, and B. Thuraisingham. Deepsweep: An evaluation framework for mitigating dnn backdoor attacks using data augmentation. *arXiv:2012.07006*, 2020.

[308] Y. Zeng, S. Chen, W. Park, Z. Mao, M. Jin, and R. Jia. Adversarial unlearning of backdoors via implicit hypergradient. In *International Conference on Learning Representations*, 2022.

[309] C. Zhang, S. Bengio, and Y. Singer. Are all layers created equal? *CoRR*, 2019.

[310] H. Zhang, H. Chen, C. Xiao, S. Gowal, R. Stanforth, B. Li, D. Boning, and C.-J. Hsieh. Towards stable and efficient training of verifiably robust neural networks. In *International Conference on Learning Representations; ICLR*, 2020.

[311] J. Zhang, J. Chen, D. Wu, B. Chen, and S. Yu. Poisoning attack in federated learning using generative adversarial nets. In *IEEE International Conference On Trust, Security and Privacy*, pages 374–380. IEEE, 2019.

[312] J. Zhang, D. Wu, C. Liu, and B. Chen. Defending poisoning attacks in federated learning via adversarial training method. In *International Conference on Frontiers in Cyber Security*, pages 83–94. Springer, 2020.

[313] R. Zhang and Q. Zhu. A game-theoretic analysis of label flipping attacks on distributed support vector machines. In *2017 51st Annual Conf. on Inf. Sciences and Sys., CISS 2017*, pages 1–6. IEEE, 2017.

[314] T. Zhang. Multi-stage convex relaxation for learning with sparse regularization. In *Advances in Neural Information Processing Systems, NeurIPS*. Curran Associates, Inc., 2008.

[315] X. Zhang, Y. Ma, A. Singla, and X. Zhu. Adaptive reward-poisoning attacks against reinforcement learning. In *International Conference on ML, ICML 2020*, pages 11225–11234. PMLR, 2020.

[316] X. Zhang, Z. Zhang, and T. Wang. Trojaning language models for fun and profit. *CoRR*, abs/2008.00312, 2020.

[317] X. Zhang, H. Chen, and F. Koushanfar. Tad: Trigger approximation based black-box trojan detection for ai. *arXiv:2102.01815*, 2021.

[318] Z. Zhang, J. Jia, B. Wang, and N. Z. Gong. Backdoor attacks to graph neural networks. In *SACMAT '21: The 26th ACM Symposium on Access Control Models and Technologies*, pages 15–26. ACM, 2021.

[319] P. Zhao, P.-Y. Chen, P. Das, K. N. Ramamurthy, and X. Lin. Bridging mode connectivity in loss landscapes and adversarial robustness. In *International Conference on Learning Representations, ICLR 2021*, 2020.

[320] S. Zhao, X. Ma, X. Zheng, J. Bailey, J. Chen, and Y. Jiang. Clean-label backdoor attacks on video recognition models. In *IEEE/CVF International Conference on Computer Vision, ICCV 2020*, pages 14431–14440. IEEE, 2020.

[321] Y. Zhao, J. Chen, J. Zhang, D. Wu, J. Teng, and S. Yu. Pdgan: a novel poisoning defense method in federated learning using generative adversarial network. In *International Conference on Algorithms and Architectures for Parallel Processing*, pages 595–609. Springer, 2019.

[322] H. Zhong, C. Liao, A. C. Squicciarini, S. Zhu, and D. J. Miller. Backdoor embedding in convolutional neural network models via invisible perturbation. In *ACM Conference on Data and Application Security and Privacy, CODASPY*, pages 97–108. ACM, 2020.

[323] C. Zhu, W. R. Huang, H. Li, G. Taylor, C. Studer, and T. Goldstein. Transferable clean-label poisoning attacks on deep neural nets. In *International Conference on ML, ICML 2019*, pages 7614–7623. PMLR, 2019.

[324] L. Zhu, R. Ning, C. Wang, C. Xin, and H. Wu. Gangsweep: Sweep out neural backdoors by gan. In *28th ACM International Conference on Multimedia*, pages 3173–3181, 2020.

[325] L. Zhu, R. Ning, C. Xin, C. Wang, and H. Wu. Clear: Clean-up sample-targeted backdoor in neural networks. In *IEEE/CVF International Conference on Computer Vision, ICCV 2021*, pages 16453–16462, 2021.

[326] D. Zügner and S. Günnemann. Adversarial attacks on graph neural networks via meta learning. In *International Conference on Learning Representations, ICLR 2019*. OpenReview.net, 2019.

Appendix A

Understanding Backdoor Poisoning Vulnerability

In [59] we have shown the backdoor learning curves only for some classifiers. Here, we report them for all the classifiers considered in this work. As we will discuss later in this section, these results confirm the ones provided in Chapter 5. In particular, here we consider:

- Support Vector Machine (SVM) with $\lambda \in \{100, 0.1\}$ for MNIST, $\lambda \in \{10000, 0.1\}$ for CIFAR10, and $\lambda \in \{100000, 1\}$ for Imagenette.
- Ridge Classifier (RC) with $\lambda \in \{1000, 1\}$ for MNIST, $\lambda \in \{10000, 1\}$ for CIFAR10, and $\lambda \in \{100000, 1\}$ for Imagenette.
- Logistic Classifier (LC) with $\lambda \in \{10, 0.01\}$ for MNIST, $\lambda \in \{10000, 100\}$ for CIFAR10, and $\lambda \in \{100000, 10\}$ for Imagenette.
- SVM with an RBF kernel, where $\lambda \in \{1, 0.01\}$ and $\gamma = 5\text{e}{-04}$ for MNIST, $\lambda \in \{100, 1\}$ and $\gamma = 1\text{e}{-03}$ for CIFAR10, and $\lambda \in \{10, 0.1\}$ and $\gamma = 1\text{e}{-05}$ for Imagenette.

Moreover, we compare the results obtained on the class pairs considered in the paper (7 vs 1 on MNIST, *airplane vs frog* on CIFAR10 and Imagenette *tench vs truck*) with the ones obtained on different pairs.

Backdoor Learning Curves and Backdoor Learning Slope. In Fig. A.1-A.9 we report the backdoor learning curves for each classifier and dataset pair. In Fig. A.10-A.12, we report the backdoor learning slope, computed with $p = 0.1$, for all the considered classifiers and all subset pairs. The results do not show significant variation with respect to the ones reported in the paper.

Empirical Parameter Deviation Plots. In Fig. A.13-A.15, shows how the classifiers' parameters change when the classifiers learn the backdoors. This analysis is carried out with $p = 0.1$. The results do not vary significantly across different classifiers and class pairs. The only exception is MNIST 5 vs 2. The untainted classifier is already quite complex; therefore, it does not increase its complexity when it learns the backdoor.

Increasing the trigger size or visibility Although it is a known result in the literature that the size of the trigger increases the effectiveness of the attack [229; 230], here, for the first time to the best of our knowledge, we show how it interacts with other factors. In this section we report further experimental results when increasing the trigger size or visibility. As expected, the results in Fig. A.16-A.17 show that choosing a larger trigger enhances the effectiveness of the attack. Indeed, when the trigger is larger or more visible the backdoor learning curves go down faster. Using the proposed backdoor slope to analyze the effect of complexity, controlled via the hyperparameters, on the vulnerability against backdoors, we found a region of the hyperparameter space that leads to having desirable performances: an accuracy high on the clean test samples and low on the ones containing the backdoor trigger.

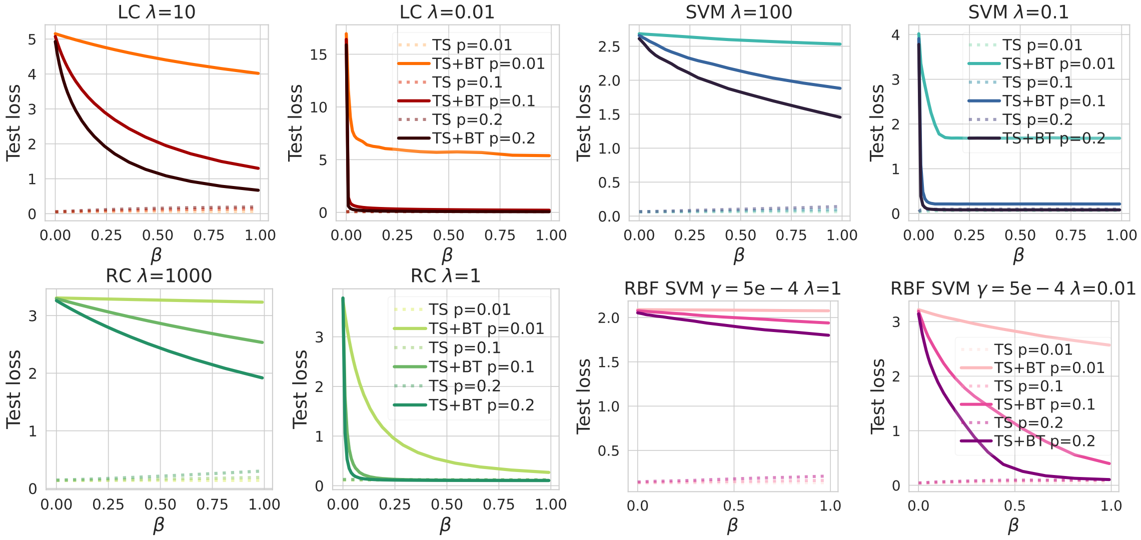


Fig. A.1: *Backdoor learning curves for different classifiers trained on MNIST 7-1. Darker lines represent a higher fraction of poisoning samples p injected into the training set. We report the loss on the clean test samples (TS) with a dashed line and on the test samples with the backdoor trigger (TS+BT) with a solid line.*

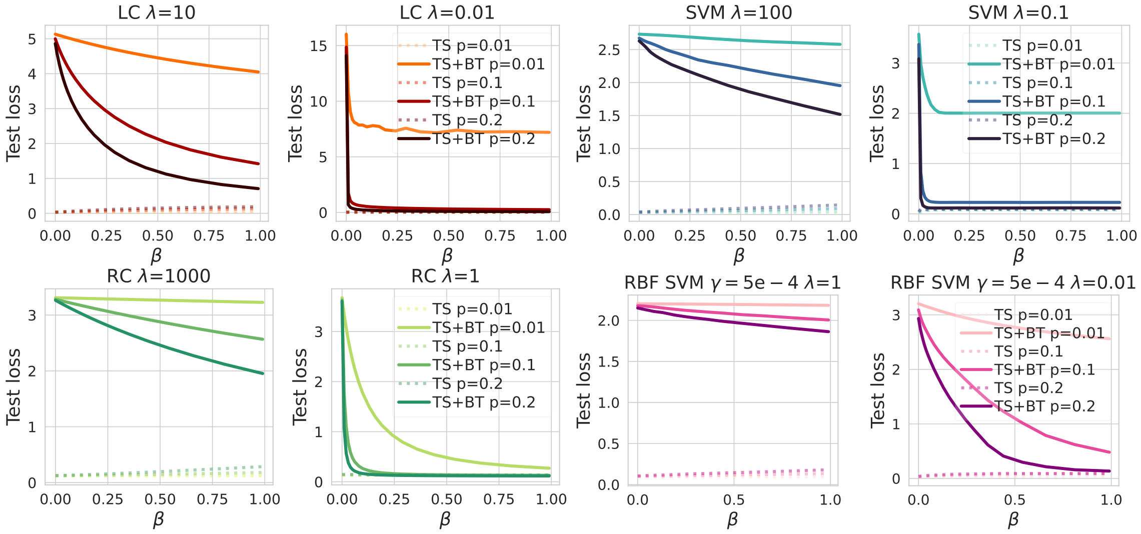


Fig. A.2: *Backdoor learning curves for different classifiers trained on MNIST 3-0. See the caption of Fig. A.1 for further details.*

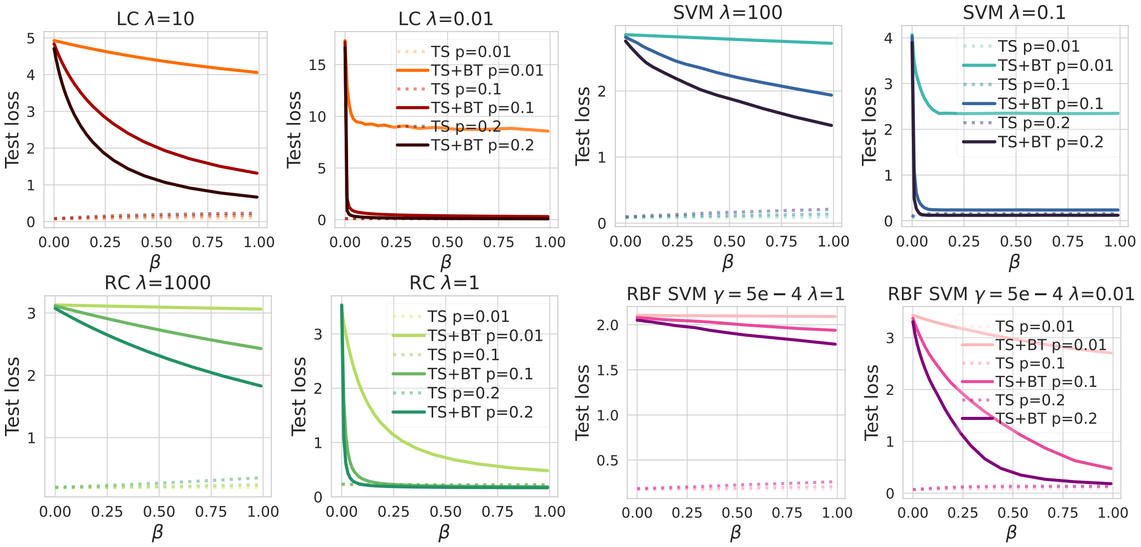


Fig. A.3: *Backdoor learning curves for different classifiers trained on MNIST 5-2. See the caption of Fig. A.1 for further details.*

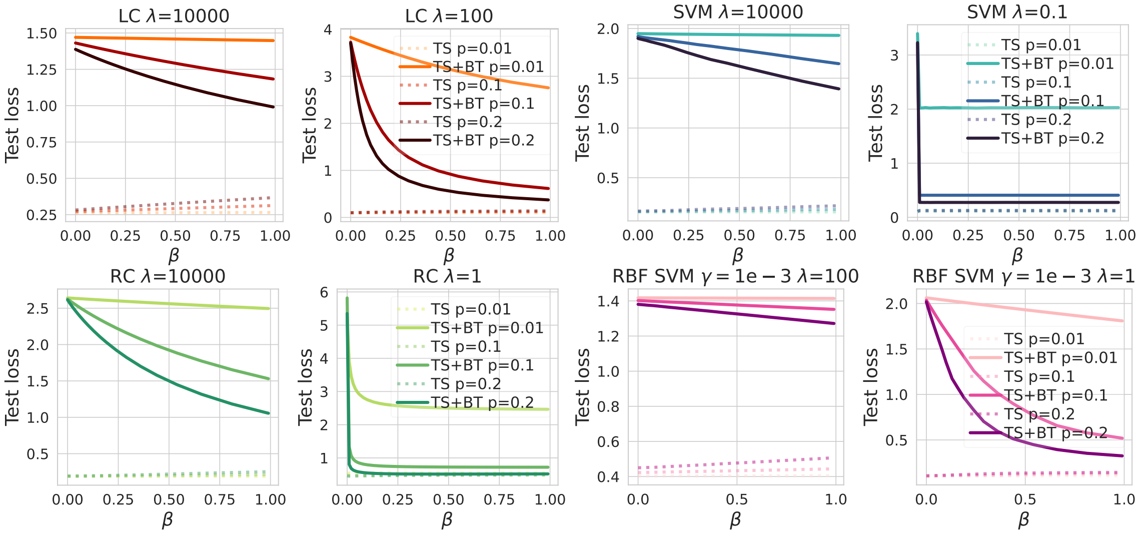


Fig. A.4: *Backdoor learning curves for different classifiers trained on CIFAR10 airplane vs frog. See the caption of Fig. A.1 for further details.*

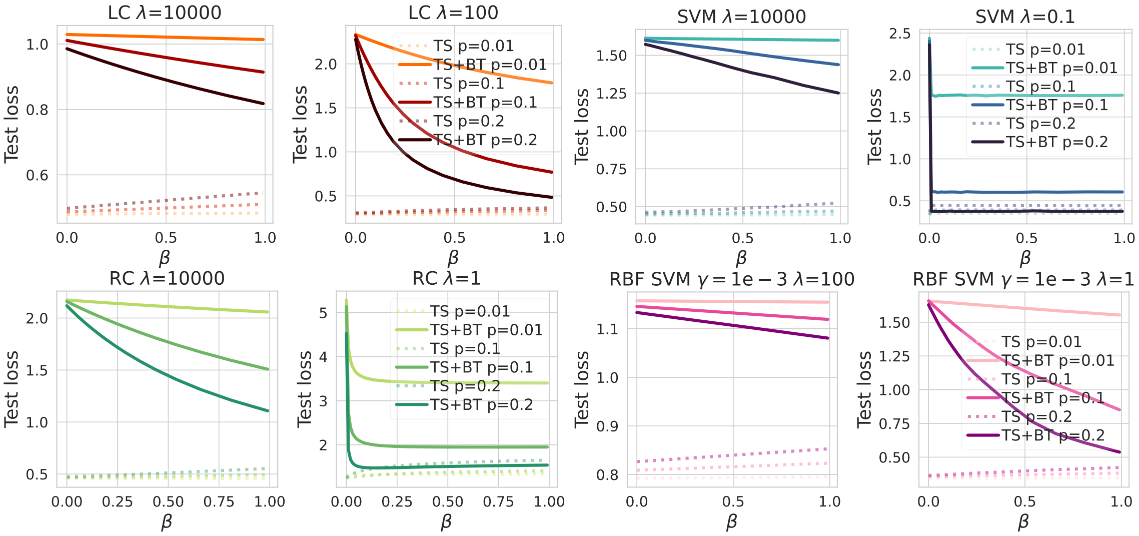


Fig. A.5: *Backdoor learning curves for different classifiers trained on CIFAR10 bird vs dog. See the caption of Fig. A.1 for further details.*

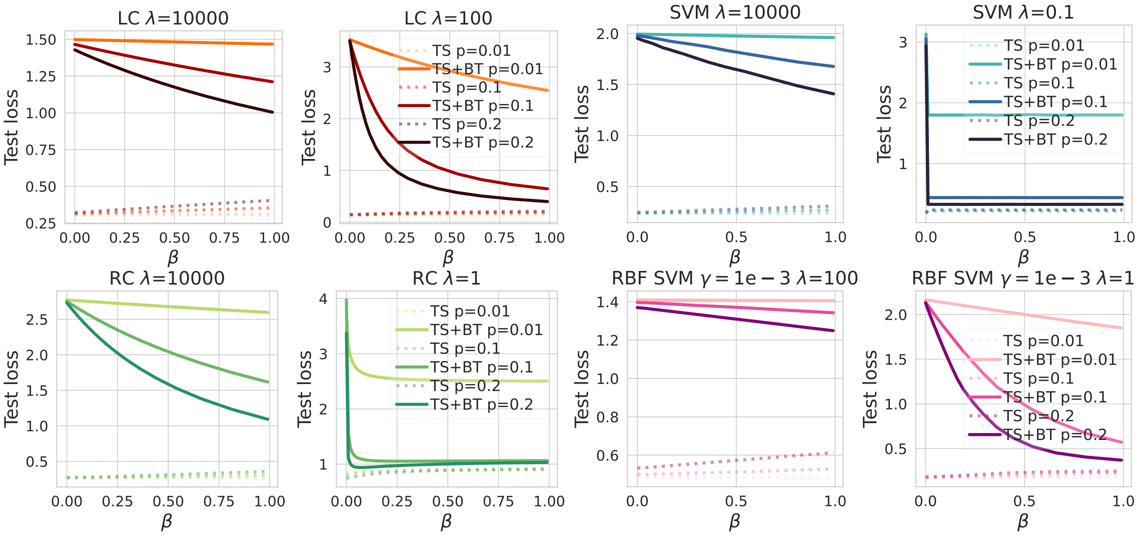


Fig. A.6: *Backdoor learning curves for different classifiers trained on CIFAR10 airplane vs truck. See the caption of Fig. A.1 for further details.*

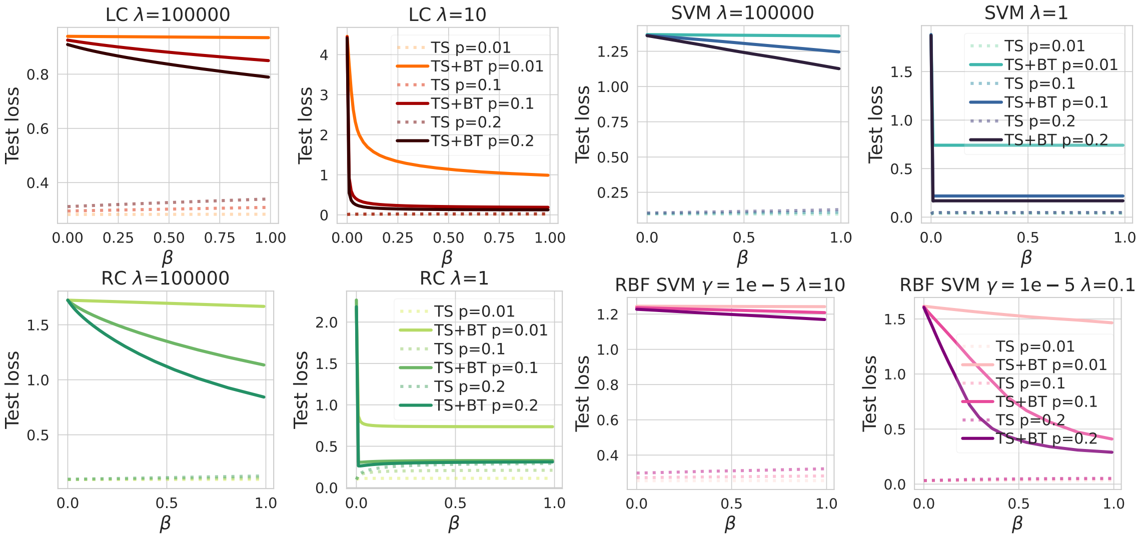


Fig. A.7: *Backdoor learning curves for different classifiers trained on Imagenette tench vs truck. See the caption of Fig. A.1 for further details.*

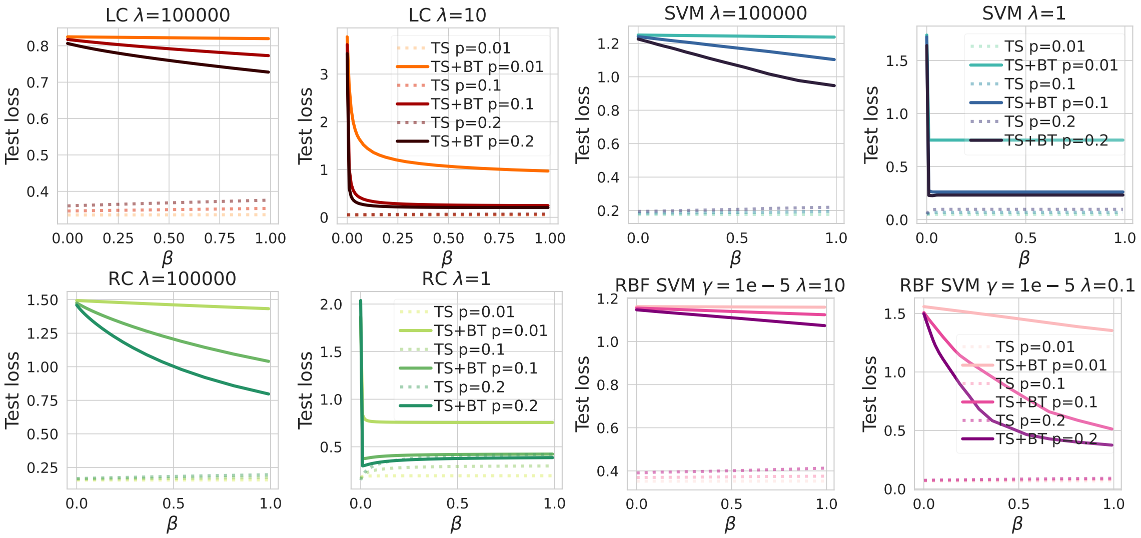


Fig. A.8: *Backdoor learning curves for different classifiers trained on Imagenette cassette player vs church. See the caption of Fig. A.1 for further details.*

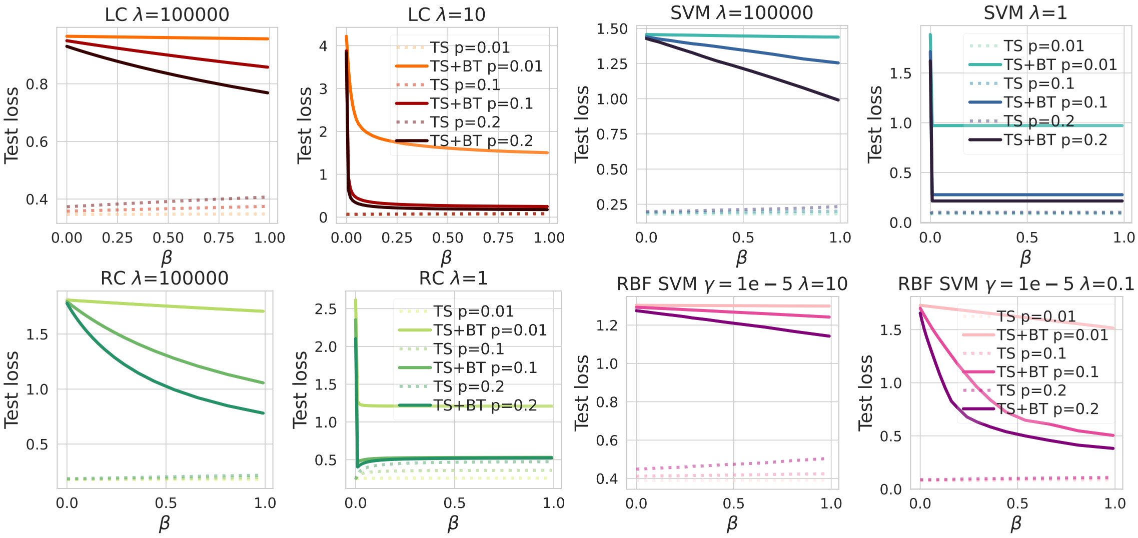


Fig. A.9: *Backdoor learning curves for different classifiers trained on Imagenette tench vs parachute. See the caption of Fig. A.1 for further details.*

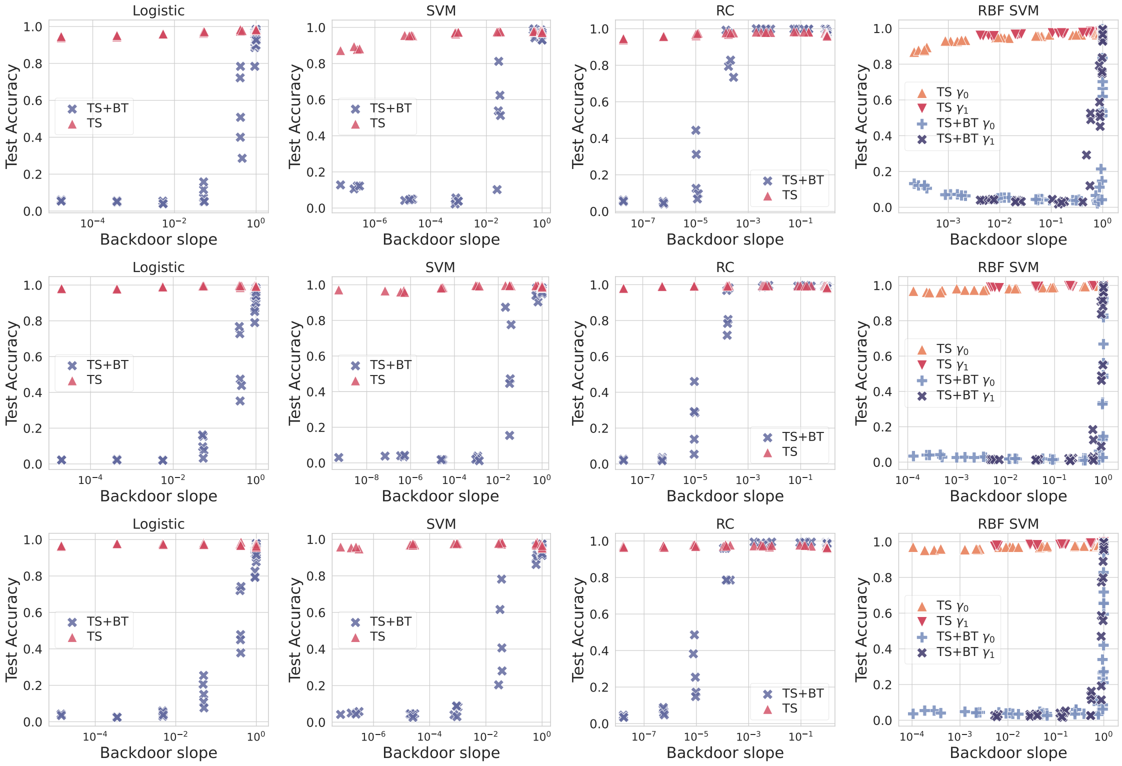


Fig. A.10: *Backdoor slope* η *vs clean accuracy (red) and backdoor effectiveness (blue) on MNIST 7vs.1 (top row), 3vs.0 (middle row) and 5vs.2 (bottom row). We measure the classification accuracy on the untainted test samples (TS), and on the same samples after adding the* 3×3 *backdoor trigger (TS+BT). We chose the* γ *parameter for the RBF kernel as* $\gamma_0 = 5\mathrm{e}{-04}$ *(orange triangle for clean data, light blue plus for data with trigger) and* $\gamma_1 = 5\mathrm{e}{-03}$ *(red inverted triangle for clean data, dark blue x for data with trigger).*

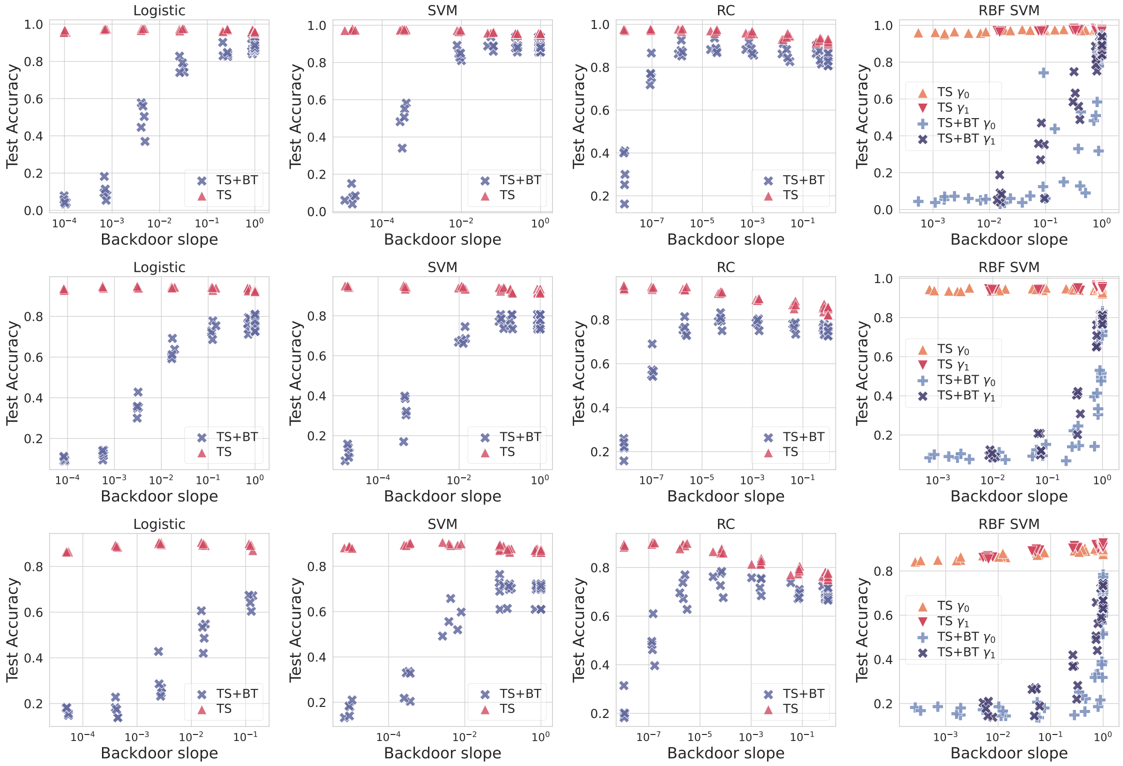


Fig. A.11: *Backdoor slope vs backdoor (BK) effectiveness on CIFAR10 airplane vs frog (top row), airplane vs truck (middle row) and bird vs dog (bottom row). See the caption of Fig. A.10 for further details. The results are obtained considering a trigger size equal to 8.*

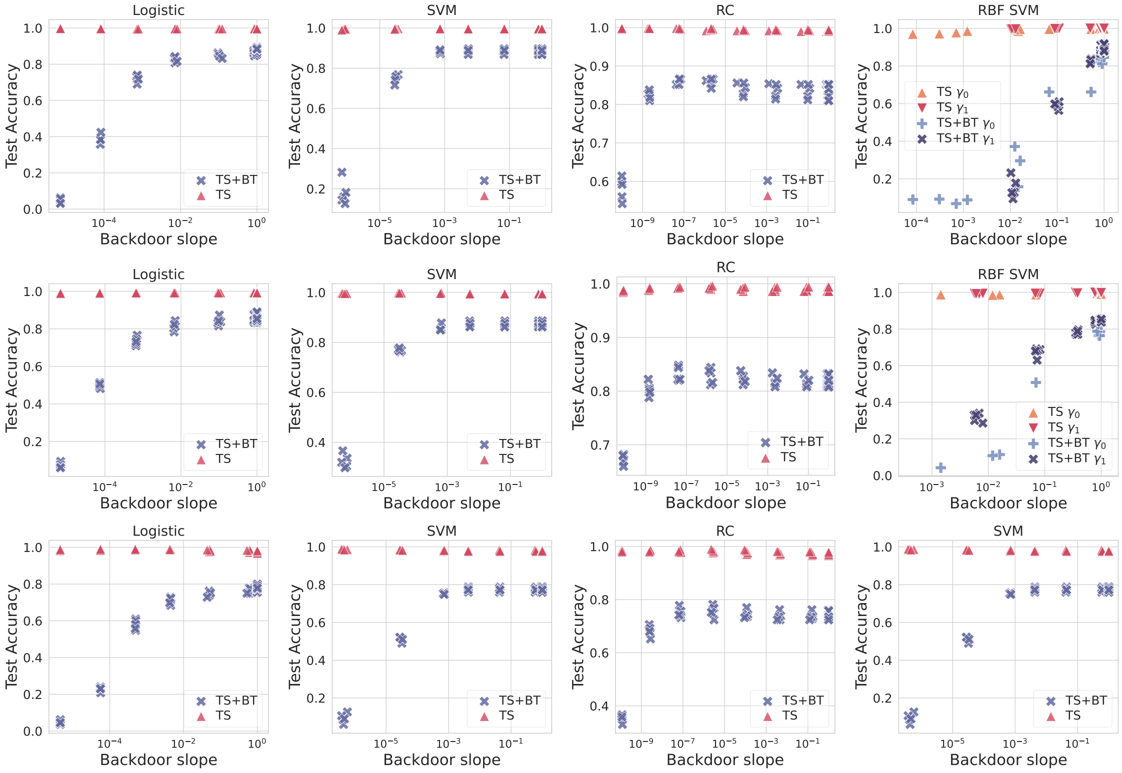


Fig. A.12: *Backdoor slope vs backdoor (BK) effectiveness on Imagenette tench vs truck (top row), cassette player vs church (middle row) and tench vs parachute (bottom row). See the caption of Fig. A.10 for further details. The results are obtained considering a trigger size equal to 8.*

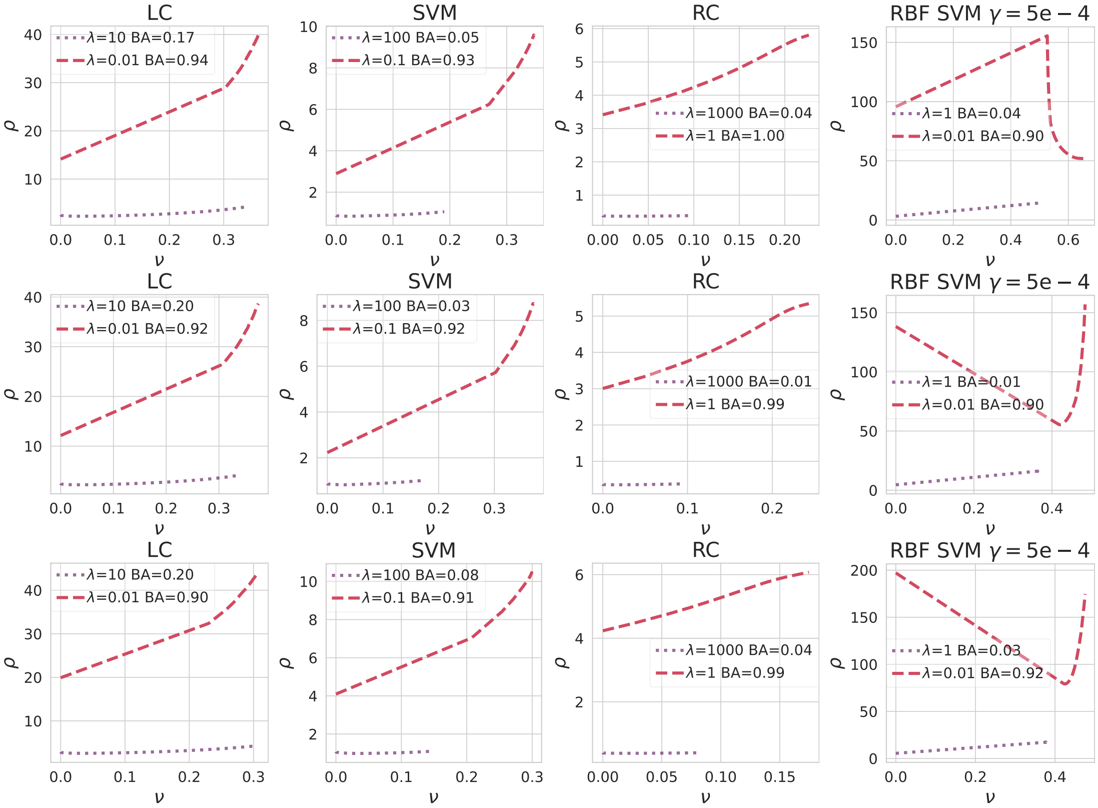


Fig. A.13: *Backdoor weight deviation for different classifiers trained on MNIST* 7 vs 1 *(top row),* 3 vs 0 *(middle row) and* 5 vs 2 *(bottom row). We specify regularization parameter* λ *and backdoor (BK) accuracy for each setting in the legend of each plot.*

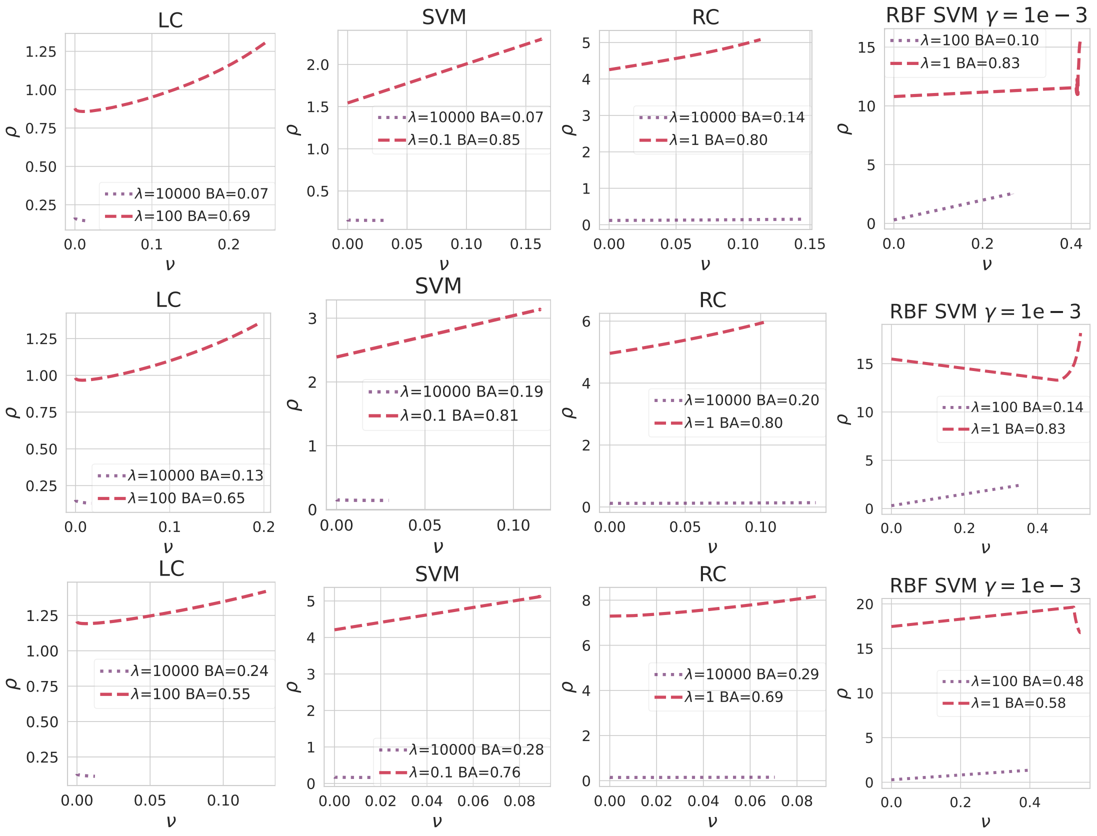


Fig. A.14: *Backdoor weight deviation for different classifiers trained on CIFAR10 airplane vs frog (top), airplane vs truck (middle), and bird vs dog (bottom). See Fig. A.13 for further details.*

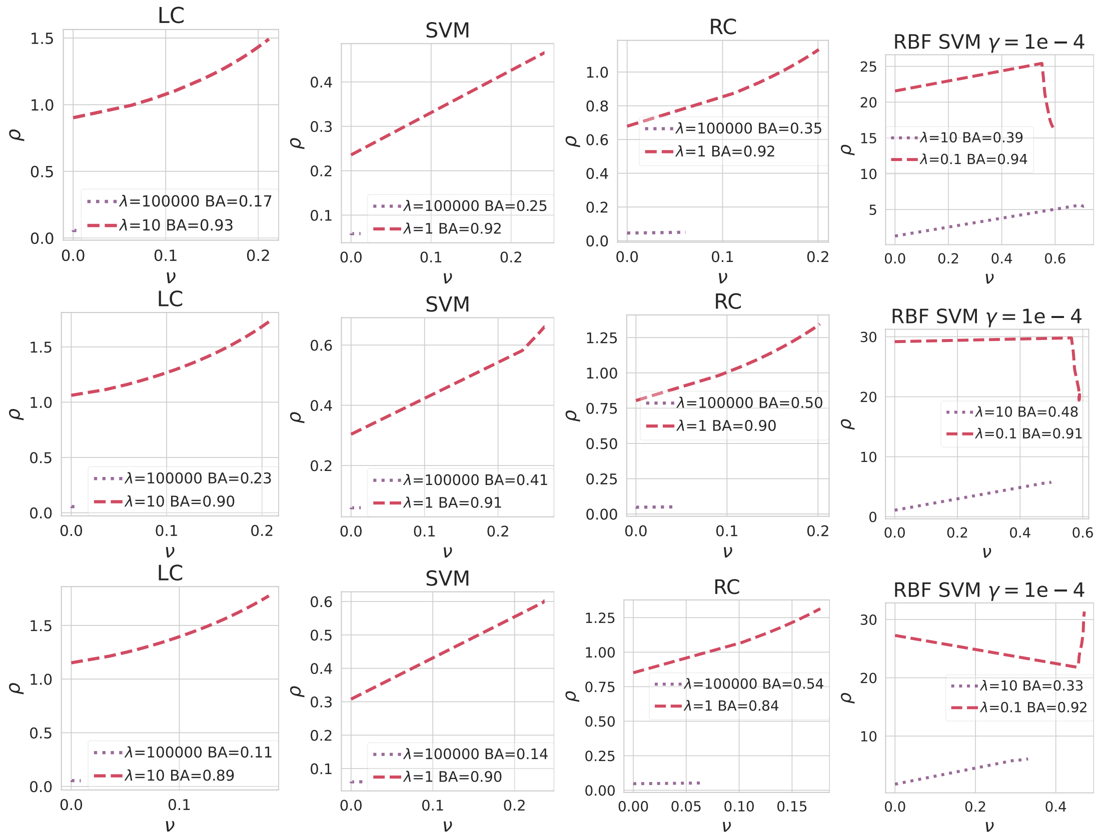
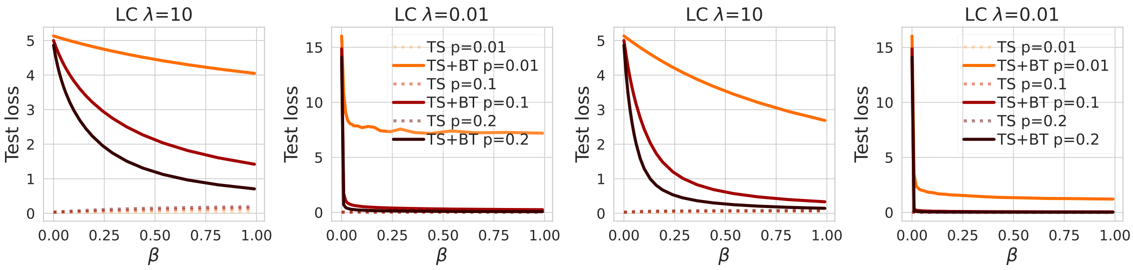
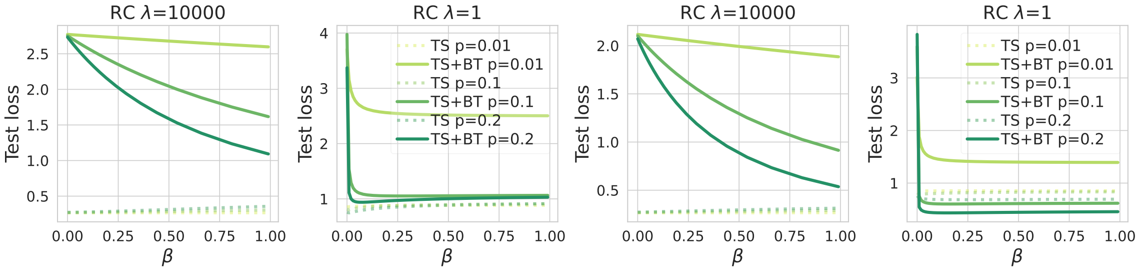


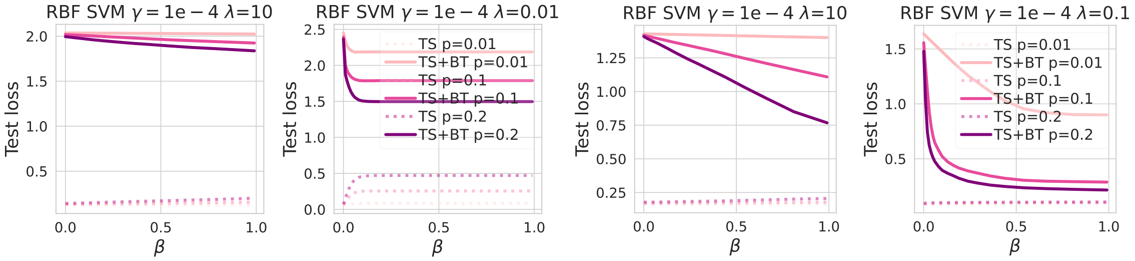
Fig. A.15: *Backdoor weight deviation for different classifiers trained on Imagenette tench vs truck (top), tench vs parachute (middle), and cassette player vs church (bottom). See Fig. A.13 for further details.*



(a) *MNIST trigger size* 3×3. **(b)** *MNIST trigger size* 6×6.

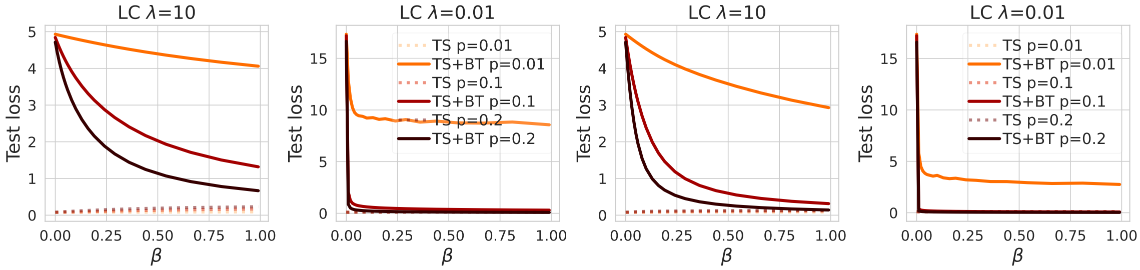


(c) *CIFAR10 trigger size* 8×8. **(d)** *CIFAR10 trigger size* 16×16.

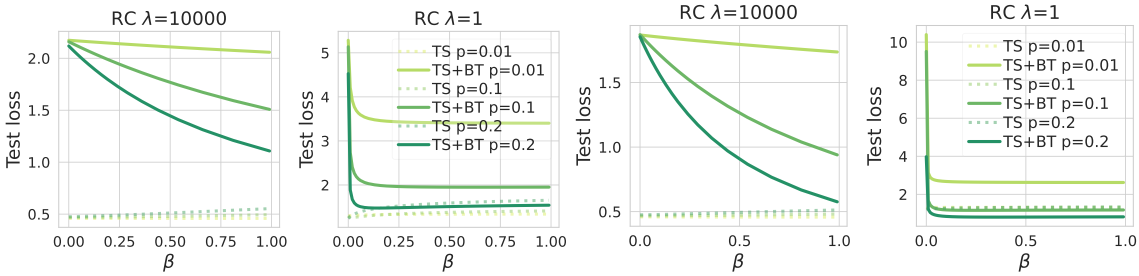


(e) *Imagenette trigger visibility* $c_m = 10$. **(f)** *Imagenette trigger visibility* $c_m = 75$.

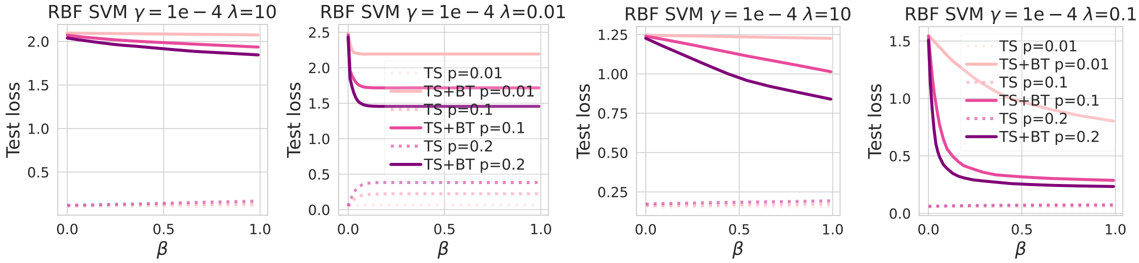
Fig. A.16: *Backdoor learning curves for: (top row) LC on MNIST 3vs.0 with trigger size 3×3 (left) or 6×6 (right); (middle row) RC on CIFAR10 airplane vs truck with trigger size 8×8 (left) or 16×16 (right); (bottom row) RBF SVM on Imagenette cassette player vs church with trigger visibility* c_m*=10 (left) or* c_m*=75 (right). Further details in Fig. A.1.*



(a) *MNIST trigger size* 3×3. **(b)** *MNIST trigger size* 6×6.



(c) *CIFAR10 trigger size* 8×8. **(d)** *CIFAR10 trigger size* 16×16.



(e) *Imagenette trigger visibility* $c_m = 10$. **(f)** *Imagenette trigger visibility* $c_m = 75$.

Fig. A.17: *Backdoor learning curves for: (top row) LC on MNIST 5 vs. 2; (middle row) RC on CIFAR10 bird vs dog; (bottom row) RBF SVM on Imagenette tench vs parachute.*

Appendix B

Causing Energy-Latency Failures via Poisoning

Layers Activation on CIFAR10. We depict in Fig. B.1 the layer's activations for clean and sponge ResNet18 and VGG16 trained on CIFAR10. Notably, the results are consistent with those presented for GTSRB and Celeb in the paper. Indeed, the modules mostly affected by the sponge attack are ReLu and MaxPooling. We further note that for ResNet18, the ReLu operators placed at the beginning have a greater impact than those placed at the end. For example, in first ResNet18 ReLu in Fig. B.1 increases the number of activations by about 40% (i.e., from 50% to 90%), while the last ReLu has an increase of only 10%. However, this phenomenon is not observed on the Celeb dataset, where all the ReLu are largely affected, suggesting that increasing data dimensionality may enhance the sponge's effectiveness.

Attack Hyperparameters Tuning. In Fig. B.2 we show the effect of the two hyperparameters λ and σ on the test sets of the three dataset considered in our paper. The top row contains the same results proposed in the main paper, using a validation set with 100 samples, i.e., considering an attacker with a low budget for tuning the attack's hyperparameter. We proposed the same ablation analysis in the bottom row while considering all the test samples, i.e., more than $10,000$, for each dataset. We can observe that the results of the two analyses are almost the same, meaning that using a small validation set to tune the attacks' hyperparameter does not lead to suboptimal choices, which makes our attack more feasible in realistic scenarios where gathering data is costly.

Increasing Attacker's Budget In Fig. B.3 we report the energy increase when the percentage of poisoning gradient update p grows. We note that our attack can also succeed when manipulating a few gradient updates during model training. This property allows our attack to be applicable even in other contexts, such as federated learning, where the attackers can usually compromise only few nodes.

Reversing Sponge Models In Sec. 11 we derived a novel defensive strategy to mitigate the effect of a sponge poisoning attack. Given a sponge model, the victim user tries to fine-tune it to minimize the energy consumption while preserving the model's accuracy (Eq. 6.5). To this end, we adapt the training algorithm proposed in Alg. 5, used to stage a poisoning attack, to meet the novel sanitization objective function. We fine-tune the sponge models on CIFAR10, GTSRB, and CelebA for 100 training epochs with SGD optimizer with momentum 0.9, weight decay $5e-4$, p equals to 0.05, and batch size 512, optimizing the cross-entropy loss. We employ an exponential learning scheduler with an initial learning rate of 0.025 and decay of 0.95. As for sponge attack, we analyze the effect of hyperparameters λ and σ on sanitized models, looking for the best configurations that enable high prediction accuracy and low energy consumption. The obtained results are reported in Fig. B.4 and Fig. B.5 respectively for σ and λ. Regarding the former, we observe that high values tend to clip to zero more activations, thus decreasing the energy consumption. Conversely, for

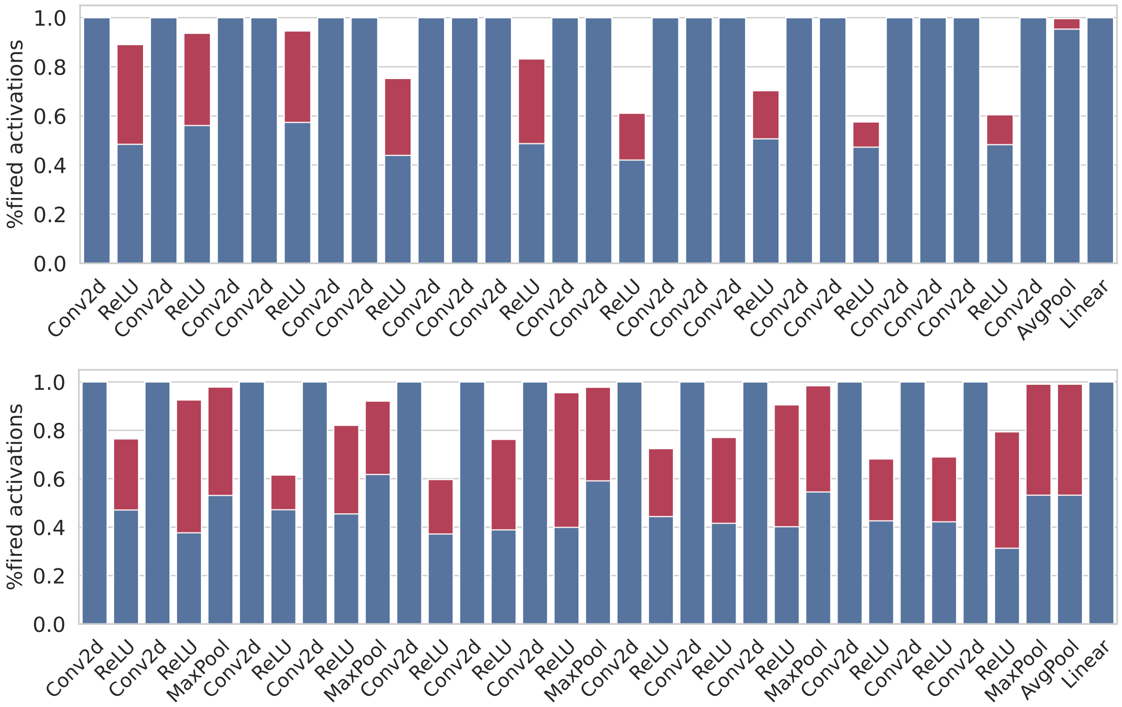


Fig. B.1: *Layers activations for ResNet18 (top) and VGG16 (bottom) in CIFAR10.*

very small values of σ, as for the sponge attack, the training algorithm becomes unstable as the $\hat{\ell}_0$ may not be sufficiently smooth to facilitate the optimization. For the latter, we observe that high values of λ tend to give excessive relevance to the energy minimization component regardless of the model's accuracy. Indeed, when increasing λ we obtain energy-efficient models, even better than a standard training algorithm, but useless as their validation or test accuracy is poor. Considering both Fig. B.4 and Fig. B.5, we can observe that the sanitization effect tend to decrease the energy consumption, satisfying the victim objective. However, a severe reduction in energy consumption may induce the model to freeze or inactivate its neurons since they are likely to output 0. In Table 6.4 we considered the best configurations that reach low energy consumption and a high validation accuracy regime with fewer epochs.

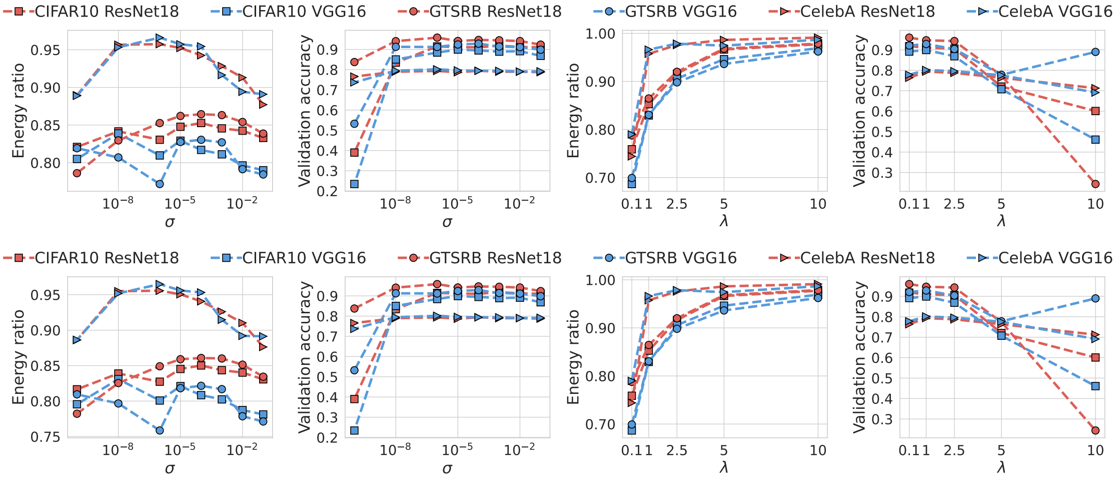


Fig. B.2: *Ablation study on σ and λ when the validation set size is 100 (top) or equal to the test test size (bottom). When analyzing λ we consider the σ value, which gives the highest energy consumption and does not decrease the validation accuracy.*

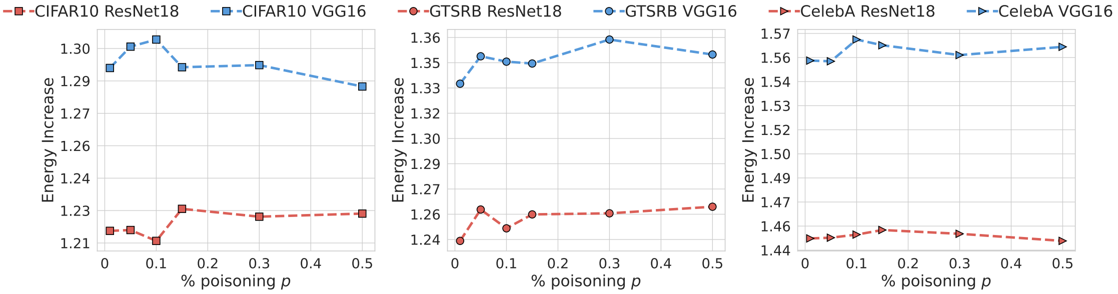


Fig. B.3: *Ablation study on the percentage of poisonign p.*

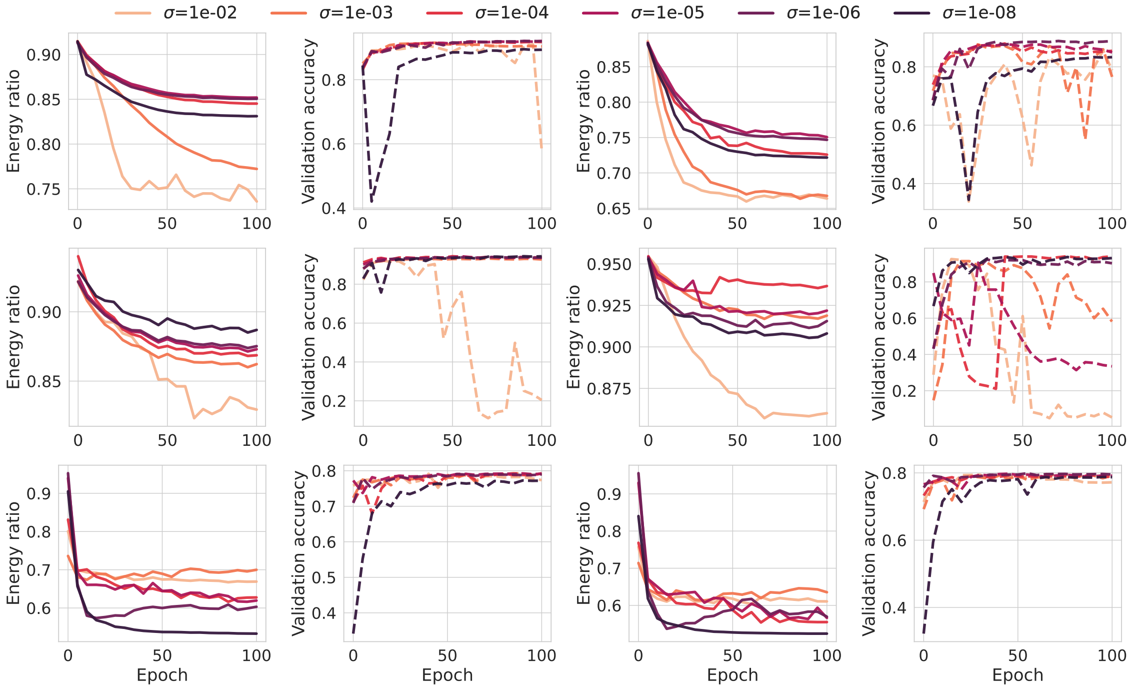


Fig. B.4: *Sponge sanitization ablation study on σ for ResNet18 (two plots on the left) and VGG16 (two plots on the right) trained on CIFAR10 (top), GTSRB (middle), and CelebA (bottom).*

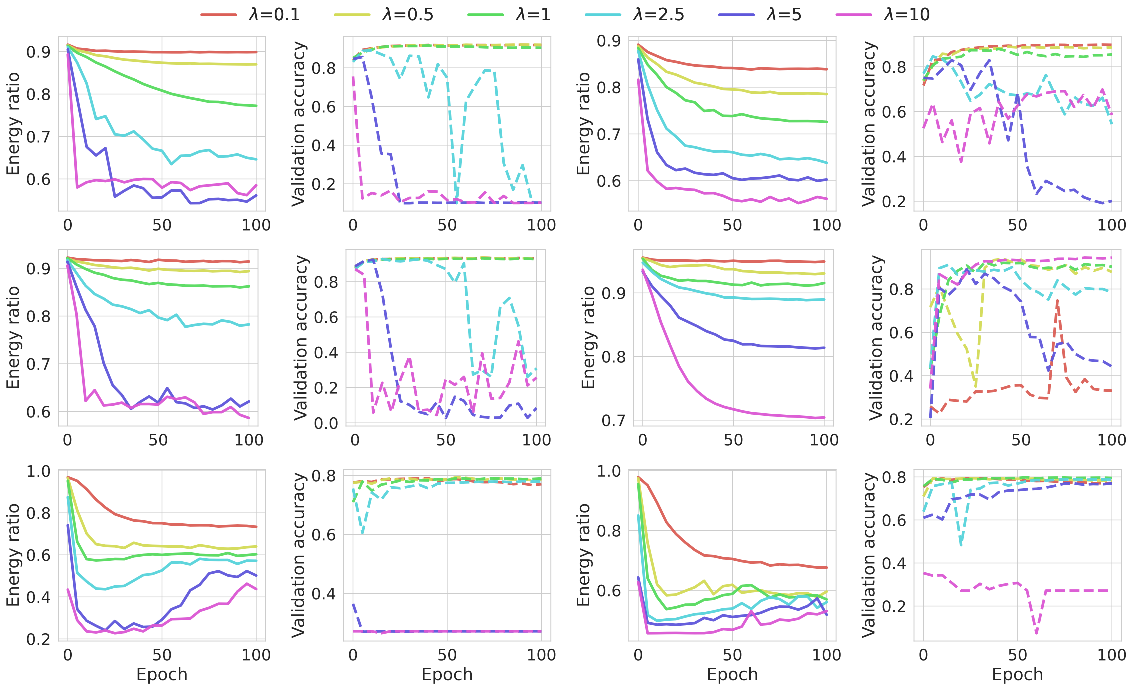


Fig. B.5: *Sponge sanitization ablation study on λ for ResNet18 (two plots on the left) and VGG16 (two plots on the right) trained on CIFAR10 (top), GTSRB (middle), and CelebA (bottom).*

Appendix C

Poisoning Under Limited Knowledge

For the sake of completeness and reproducibility of the experimental setting, in the following, we report a detailed list of all hyperparameters used in our experiments. In particular Table C.1 and C.2 present all the hyperparameters used in our method for the comparison with [55]. Table C.3 contains the hyperparameters used by our algorithm during the comparison.

Method	G	λ	p_c	p_m	p_z
Ours ($\delta = 73.43$)	150	1	0.85	0.2	0.35
Ours ($\delta = 146.87$)	150	1	0.85	0.01	0.20
Ours ($\delta = \infty$)	150	1	0.85	0.001	0.25

Method	G	λ	p_c	p_m	p_z
Ours ($\delta = 73.43$)	150	1	0.85	0.2	0.35
Ours ($\delta = 146.87$)	150	1	0.85	0.01	0.20
Ours ($\delta = \infty$)	150	1	0.85	0.001	0.25

Table C.1: *Comparison parameters for Digits dataset with digits 8&9 (left) and 4&1 (right).*

Method	G	λ	p_c	p_m	p_z
Ours ($\delta = 73.43$)	150	1	0.85	0.015	0.10
Ours ($\delta = 146.87$)	150	1	0.85	0.015	0.25
Ours ($\delta = \infty$)	150	1	0.85	0.005	0.25

Method	G	λ	p_c	p_m	p_z
Ours ($\delta = 73.43$)	150	1	0.85	0.02	0.05
Ours ($\delta = 146.87$)	150	1	0.85	0.01	0.10
Ours ($\delta = \infty$)	150	1	0.85	0.001	0.15

Table C.2: *Comparison parameters for MNIST dataset with digits 3&2 (left) and 1&4 (right).*

Method	G	λ	p_c	p_m	p_z
Ours ($\delta = 0.15$)	20	1	0.85	0.01	0.10
Ours ($\delta = 0.30$)	20	1	0.85	0.01	0.10

Method	G	λ	p_c	p_m	p_z
Ours ($\delta = 10$)	50	1	0.85	0.15	0.20
Ours ($\delta = 20$)	50	1	0.85	0.15	0.20

Table C.3: *Comparison parameters for Seeds (left) and MoCap Hand Postures (right).*

Università
Ca’Foscari
Venezia

DEPOSITO ELETTRONICO DELLA TESI DI DOTTORATO

DICHIARAZIONE SOSTITUTIVA DELL’ATTO DI NOTORIETA’

(Art. 47 D.P.R. 445 del 28/12/2000 e relative modifiche)

Io sottoscritto ……Antonio Emanuele Cinà……………………………………………

nato … a Palermo……………………………………………. (prov. PA……) il ……04/10/1995

residente a ……Povegliano……………………… in …Via Borè………………………… n. …9…

Matricola (se posseduta) ………854866………….. Autore della tesi di dottorato dal titolo:

………………Vulnerability of Machine Learning: A Study on Poisoning Attacks…………

…………………………………………………………………………………………………

…………………………………………………………………………………………………

Dottorato di ricerca in …………Informatica…………………………….

(in cotutela con …………………………………………………………………………………)

Ciclo ……35………………

Anno di conseguimento del titolo ……………2023………

DICHIARO

di essere a conoscenza:
1) del fatto che in caso di dichiarazioni mendaci, oltre alle sanzioni previste dal codice penale e dalle Leggi speciali per l’ipotesi di falsità in atti ed uso di atti falsi, decado fin dall’inizio e senza necessità di nessuna formalità dai benefici conseguenti al provvedimento emanato sulla base di tali dichiarazioni;
2) dell’obbligo per l’Università di provvedere, per via telematica, al deposito di legge delle tesi di dottorato presso le Biblioteche Nazionali Centrali di Roma e di Firenze al fine di assicurarne la conservazione e la consultabilità da parte di terzi;
3) che l’Università si riserva i diritti di riproduzione per scopi didattici, con citazione della fonte;
4) del fatto che il testo integrale della tesi di dottorato di cui alla presente dichiarazione viene archiviato e reso consultabile via internet attraverso l’Archivio Istituzionale ad Accesso Aperto dell’Università Ca’ Foscari, oltre che attraverso i cataloghi delle Biblioteche Nazionali Centrali di Roma e Firenze;
5) del fatto che, ai sensi e per gli effetti di cui al D.Lgs. n. 196/2003, i dati personali raccolti saranno trattati, anche con strumenti informatici, esclusivamente nell’ambito del procedimento per il quale la presentazione viene resa;
6) del fatto che la copia della tesi in formato elettronico depositato nell’Archivio Istituzionale ad Accesso Aperto è del tutto corrispondente alla tesi in formato cartaceo, controfirmata dal tutor, consegnata presso la segreteria didattica del dipartimento di riferimento del corso di dottorato ai fini del deposito presso l’Archivio di Ateneo, e che di conseguenza va esclusa qualsiasi responsabilità dell’Ateneo stesso per quanto riguarda eventuali errori, imprecisioni o omissioni nei contenuti della tesi;
7) del fatto che la copia consegnata in formato cartaceo, controfirmata dal tutor, depositata nell’Archivio di Ateneo, è l’unica alla quale farà riferimento l’Università per rilasciare, a richiesta, la dichiarazione di conformità di eventuali copie.

Data _03/12/2022________ **Firma ________________________**

Estratto per riassunto della tesi di dottorato

L'estratto (max. 1000 battute) deve essere redatto sia in lingua italiana che in lingua inglese e nella lingua straniera eventualmente indicata dal Collegio dei docenti.

L'estratto va firmato e rilegato come ultimo foglio della tesi.

Studente: Antonio Emanuele Cinà **matricola**: 854866

Dottorato: Informatica

Ciclo: 35

Titolo della tesi[1] **:** Vulnerability of Machine Learning: A Study on Poisoning Attacks

Abstract: Il successo senza precedenti dell'apprendimento automatico (ML) in diverse applicazioni è stato intrinsecamente favorito dalla crescente disponibilità di potenza di calcolo e di grandi insiemi di dati di addestramento, con l'ipotesi implicita che tali insiemi di dati siano ben rappresentativi dei dati che si incontreranno al momento del test. Tuttavia, questo presupposto può essere violato in presenza di attacchi di tipo poisoning, che presuppongono la capacità di un utente malevolo di poter compromettere i dati di addestramento o ottenere un certo controllo sul processo di apprendimento (ad esempio, quando l'addestramento del modello viene affidato a un servizio di terze parti non affidabile) al fine di degradare le prestazioni del modello al momento del test. Un attento monitoraggio delle procedure di raccolta dei dati e di addestramento dei modelli sta dunque diventando imperativo, soprattutto dopo gli ultimi incidenti in applicazioni di ML nel mondo reale causati da questo tipo di attacchi. Data la loro rilevanza pratica, sono stati pubblicati diversi articoli scientifici sugli attacchi al tempo di addestramento contro i modelli di ML. Tuttavia, nonostante l'enorme interesse suscitato da questo argomento, abbiamo riscontrato molta confusione, idee sbagliate e questioni aperte che indaghiamo in questa tesi. Affrontiamo quindi 5 diverse domande di ricerca, ovvero: (1) come classificare gli attacchi di avvelenamento; (2) come renderli scalabili nella pratica (3) come analizzarli e comprendere i fattori che influenzano la loro efficacia contro i modelli di ML; (4) come l'avvelenamento può influenzare altri aspetti di ML, andando oltre le violazioni di misclassificazione; e (5) come un attaccante può creare campioni di avvelenamento quando ha accesso al sistema solo attraverso queries. Per ognuna di queste domande di ricerca, rivediamo il problema di fondo nell'affrontare tale domanda, il corrispondente stato dell'arte in quella direzione di ricerca ed esaminiamo i contributi proposti dall'autore di questa tesi per rispondere ad essa. Infine, facciamo luce sulle limitazioni attuali e sulle domande di ricerca aperte in questo campo di ricerca e proponiamo possibili direzioni di ricerca future per affrontarle. I risultati di questa tesi aiuteranno la comunità ML a valutare meglio le potenziali vulnerabilità provenienti da dati e servizi di formazione di terze parti non affidabili. L'obiettivo è quello di aiutare a comprendere meglio tali vulnerabilità nei sistemi di ML, e a migliorarli anche in base alle nuove normative governative.

Firma dello studente

---

[1] Il titolo deve essere quello definitivo, uguale a quello che risulta stampato sulla copertina dell'elaborato consegnato.