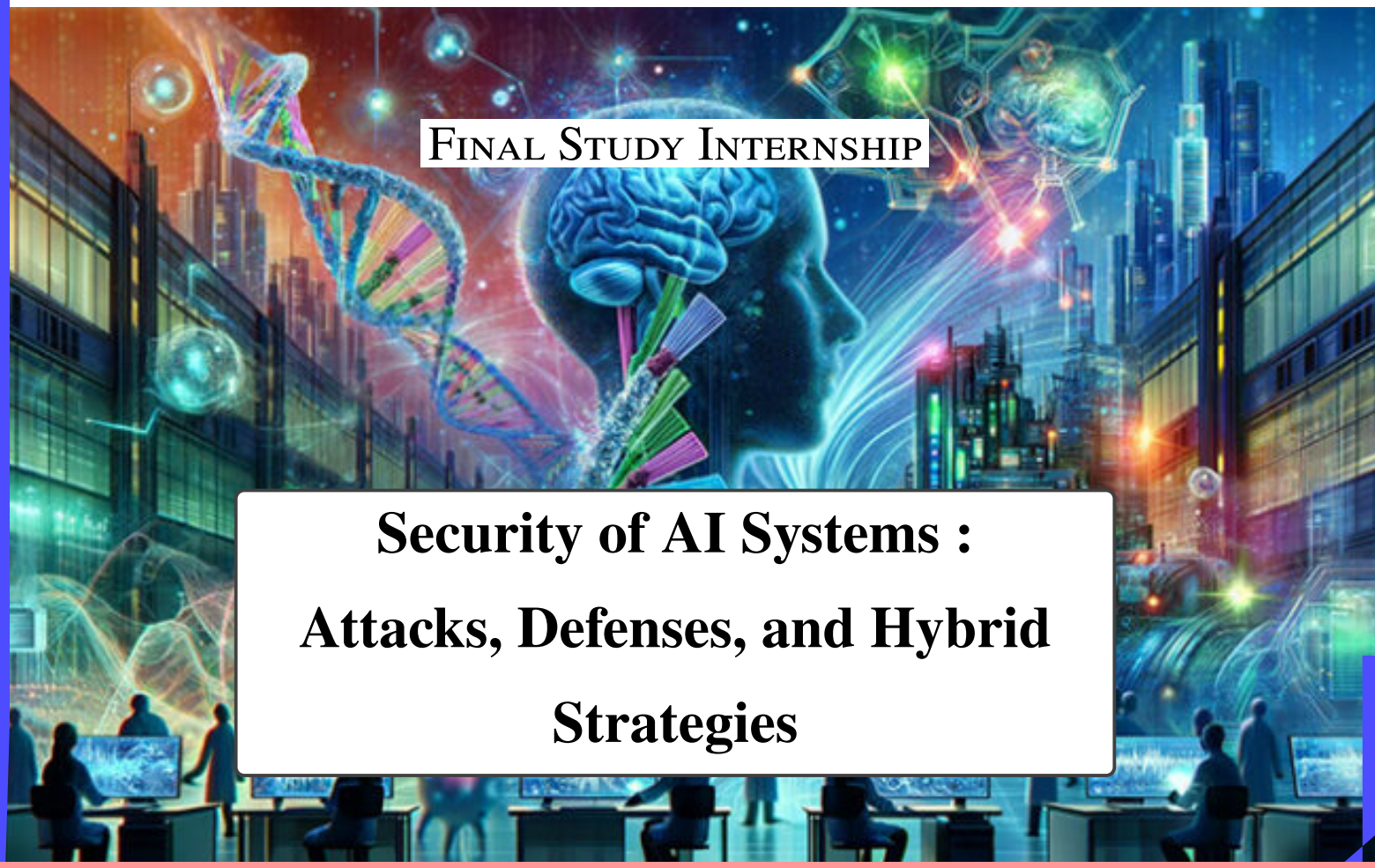
جامعة عبد المالك السعدي
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵏ ⵄⴰⴱⴷⵍⵎⴰⵍⵉⴽ ⵙⵙⴰⵄⴷⵉ
Université Abdelmalek Essaâdi

National School of Applied Sciences - Tangier

ENSA Tanger

Bio-informatics and Data Sciences

Master Thesis BISD



FINAL STUDY INTERNSHIP

Security of AI Systems : Attacks, Defenses, and Hybrid Strategies

DEFENDED ON: SEPTEMBER 9, 2025

Supervised by: **Pr. Rachida FISSOUNE**

Co-Supervised by: **Pr. Soufian BEN AMOR**

Realised By:

Oumaima ELBAHLOULI

Jury Members:

Pr. Rachida FISSOUNE	IDS Team, ENSA Tangier, Morocco	Supervisor
Pr. Asaad CHAHBOUN	ENSA Tangier, Morocco	President

Titre: Sécurité des systèmes d'IA : attaques, défenses et stratégies hybrides

Mots clés: Sécurité de l'IA, attaques adversariales, empoisonnement des données, extraction de modèle, IA en santé, défenses hybrides, COVID-19, cancer du sein, réseaux neuronaux.

Résumé: Les systèmes d'intelligence artificielle, en particulier dans des domaines sensibles tels que la santé, sont de plus en plus exposés à diverses menaces malveillantes qui compromettent leur fiabilité et leur sécurité. Ce travail présente une analyse approfondie de trois grandes catégories d'attaques visant les modèles d'IA : les attaques adversariales, qui manipulent les données d'entrée pour tromper les modèles lors de l'inférence ; l'empoisonnement des données, qui injecte des échantillons malveillants dans les jeux d'apprentissage afin de corrompre l'entraînement ; et l'extraction de modèle, qui cherche à dérober des paramètres propriétaires au moyen de requêtes stratégiques.Face à ces menaces, nous explorons et évaluons plusieurs mécanismes de défense, dont l'entraînement adversarial, la distillation défensive et des techniques de détection d'anomalies. Notre analyse expérimentale se concentre sur deux applications critiques en santé : la détection de la COVID-19 à partir de radiographies thoraciques avec des réseaux de neurones convolutifs (CNN), et le diagnostic du cancer du sein à partir de données tabulaires avec des perceptrons multicouches (MLP). Nous soumettons ces modèles à un ensemble d'attaques adversariales — telles que FGSM, PGD, BIM et Carlini-Wagner — et mesurons l'efficacité de différentes stratégies défensives.Les résultats mettent en évidence des vulnérabilités distinctes selon les modalités de données et les architectures de modèles, ce qui souligne la nécessité d'approches de défense adaptées. À partir de ces constats, nous proposons un cadre de défense hybride qui combine des méthodes de détection proactive et des améliorations de la robustesse des modèles, offrant une solution de sécurité multicouche. Ce travail contribue à l'effort global visant à développer des systèmes d'IA dignes de confiance et résilients pour la santé, garantissant à la fois la précision et la sécurité dans des applications vitales.

Title: Security of AI Systems :Attacks, Defenses, and Hybrid Strategies

Keywords: AI Security, Adversarial Attacks, Data Poisoning, Model Extraction, Healthcare AI, Hybrid Defenses, COVID-19, Breast Cancer, Neural Networks.

Abstract: Artificial intelligence systems, particularly in sensitive domains such as healthcare, are increasingly vulnerable to a variety of malicious threats that compromise their reliability and security. This research provides a comprehensive examination of three major classes of attacks against AI models: adversarial attacks, which manipulate input data to deceive models during inference; data poisoning, which injects malicious samples into training sets to corrupt learning; and model extraction attacks, which aim to steal proprietary model parameters through strategic queries. In response to these threats, we explore and evaluate multiple defense mechanisms, including adversarial training, defensive distillation, and anomaly detection techniques. Our experimental analysis focuses on two critical healthcare applications: COVID-19 detection using chest X-ray images with convolutional neural networks (CNNs), and breast cancer diagnosis using tabular data with multilayer perceptrons (MLPs). We subject these models to a suite of adversarial attacks—such as FGSM, PGD, BIM, and Carlini-Wagner—and measure the effectiveness of various defensive strategies. The results reveal distinct vulnerabilities across data modalities and model architectures, underscoring the need for tailored defense approaches. Based on these insights, we propose a hybrid defense framework that integrates proactive detection methods with model robustness enhancements, offering a multi-layered security solution. This work contributes to the broader effort to develop trustworthy and resilient AI systems for healthcare, ensuring both accuracy and security in life-critical applications.

Acknowledgements

I would like to express my deepest gratitude to my supervisors, ***Prof. Rachida FISSOUNE*** *and* ***Prof. Soufian BEN AMOR****, for their invaluable guidance, continuous support, and insightful feedback throughout this research.*

I am also grateful to the École Nationale des Sciences Appliquées (ENSA) for providing a stimulating academic environment and the resources that made this work possible.

My sincere thanks go to the members of the examination jury for their time, thoughtful comments, and constructive evaluation of this thesis.

I extend my thanks to my colleagues for their collaboration, productive discussions, and shared dedication to this research endeavor.

My heartfelt appreciation goes to my family for their constant love, patience, and understanding during this challenging yet rewarding journey.

Finally, I would like to acknowledge my own perseverance and commitment to this work, which has been both a significant academic and personal achievement.

Dedication

To my parents — for your unconditional love and the quiet strength that carried me through every challenge.

To my brothers — for your hidden love, loyalty, humor, and the constant push to aim higher.

To my sister — for your kindness, steady encouragement, and unwavering faith in me.

To my colleagues — for your collaboration, honest feedback, and shared pursuit of excellence.

To everyone who helped, loved, or cared for me, near or far — your support turned effort into achievement; this work is for you.

Contents

List of Figures

List of Tables

Chapter 1 General Introduction

Chapter Contents

1.1 Context and Motivation

Artificial intelligence (AI) has become a transformative force across numerous domains, including healthcare, finance, and autonomous systems. Its ability to process vast amounts of data, recognize patterns, and make predictions with remarkable accuracy has led to widespread adoption. However, this rapid integration of AI into critical decision-making processes has also exposed significant vulnerabilities. Malicious actors can exploit these weaknesses through adversarial attacks, data poisoning, and model extraction, threatening the reliability and safety of AI systems.

The motivation for this work stems from the urgent need to address these security challenges, particularly in healthcare, where AI-driven diagnostics and treatment recommendations directly impact patient outcomes. For instance, in medical imaging, adversarial perturbations could lead to misdiagnosis, with potentially life-threatening consequences. Similarly, in tabular data applications such as breast cancer prediction, attacks could compromise the integrity of predictive models.

Recent studies and real-world incidents have demonstrated that AI systems are highly susceptible to these threats, necessitating robust defensive mechanisms. This research is motivated by the gap between the rapid advancement of AI capabilities and the lag in developing effective security measures. By exploring both theoretical and practical aspects of AI security, this work aims to contribute to the development of trustworthy and resilient AI systems.

1.2 Definition of AI Systems

Artificial intelligence (AI) is a broad, interdisciplinary field that has garnered growing interest from researchers, industry leaders, and policymakers. Unlike many scientific domains, AI lacks a single universal definition, as its interpretation depends on context, intended applications, and the sectors involved. At its core, AI refers to computational systems designed to emulate human cognitive abilities—such as learning, reasoning, perception, and decision-making. These systems leverage techniques like machine learning, logic, and knowledge representation to operate autonomously or with human guidance, adapting through interactions with their surroundings. The book Artificial Intelligence: Definition and Background Shollo et al. (2022) underscores this diversity by presenting multiple complementary definitions, reflecting the varied ways AI can be conceptualized. From one perspective, AI involves algorithmic solutions to specific problems. Another view frames it as machines mimicking human intellectual capacities, whether basic or advanced. AI also encompasses technologies that dynamically interact with their environment, predicting outcomes and responding adaptively. A broader definition characterizes AI as autonomous systems that perceive their surroundings and take actions to achieve predefined goals—a description that captures the essence of modern AI applications.

1.3 Problem Statement and Challenges in AI Security

A minor visual modification, invisible to the human eye, can be enough to trigger a critical error in an artificial intelligence system. For instance, a slight alteration to a traffic sign could cause an autonomous vehicle to fail to recognize it, thereby compromising user safety. Such scenarios highlight the real and tangible threats facing AI, particularly in high-stakes sectors like healthcare, intelligent transportation, and cybersecurity. These systems remain vulnerable to sophisticated attacks,including malicious data injection, subtle input manipulation, the insertion of backdoors, and the leakage of sensitive information. The risk is even greater given that such attacks can occur at multiple stages of a model's lifecycle—from training to deployment—and often go unnoticed. A study by the Capgemini Research Institute Capgemini Research Institute (n.d.) found that nearly 97% of surveyed organizations experienced at least one security incident related to the use of generative AI in the past year. Beyond the technical challenges, these threats raise significant ethical and legal concerns. Key issues include the protection of personal data, transparency in algorithmic decision-making, and accountability in the event of system failure. These concerns underscore the urgent need to design more resilient AI systems, embedding security mechanisms from the early stages of development while also addressing broader societal, legal, and human considerations.

1.4 Objectives of the Project

The main objectives of this research project are as follows:

1. **Threat Analysis and Characterization**: Conduct a comprehensive study of security threats targeting artificial intelligence systems, with particular focus on adversarial attacks, data poisoning techniques, and model extraction methods. Analyze their specific implications for healthcare applications and critical decision-making systems.
2. **Defense Mechanism Evaluation**: Systematically evaluate existing defense strategies, including adversarial training, defensive distillation, and detection-based approaches, against a range of modern attacks (FGSM, PGD, BIM, Carlini-Wagner) to assess their effectiveness and limitations.
3. **Hybrid Defense Framework Development**: Design and implement an integrated hybrid defense framework that combines proactive detection mechanisms with model robustness enhancement techniques to provide comprehensive protection for AI systems across diverse data modalities.
4. **Experimental Validation on Healthcare Applications**: Conduct extensive experimental validation using two critical healthcare use cases:
    - COVID-19 detection through chest X-ray image classification using convolutional neural networks (CNNs)

- Breast cancer prediction using tabular data with multilayer perceptron (MLP) architectures

5. **Practical Guidelines and Recommendations**: Develop actionable guidelines and best practices for implementing security-by-design principles in AI systems, particularly for healthcare applications, ensuring the integration of protective measures throughout the entire AI development lifecycle.
6. **Performance Benchmarking**: Establish performance benchmarks for security and robustness metrics in medical AI systems, providing a baseline for future research and development in the field of AI security.

Through the achievement of these objectives, this project aims to make significant contributions to the field of AI security while promoting the development of safe, reliable, and trustworthy artificial intelligence systems for healthcare and other critical domains.

1.5 Organization of the Report

This report is organized to progress from context to practice and, finally, to insights. Section 2 surveys the state of the art: it introduces threat types against AI systems (adversarial examples, data poisoning, and model extraction), analyzes why modern models are vulnerable, and reviews key defense families, concluding with an overview diagram that maps attacks to countermeasures. Section 3 describes the technologies and tools used in our experiments, including runtime environment, data handling, training stack, robustness tooling, and library ecosystem. Section 4 details the experimental methodology—datasets and preprocessing (WDBC and COVID–19 Radiography), model architectures (MLP and CNN), baseline training/evaluation, the attack protocols, and the defense procedures. Section 5 presents results and analysis for each scenario: baseline behavior, Mixed Adversarial Training, Defensive Distillation, Random Noise Injection, adversarial–example detection, and the hybrid defense combining these components. Section 6 offers a general discussion that synthesizes findings across settings. Section 7 summarizes contributions and limitations. Section 8 concludes and outlines directions for future work.

Chapter 2 State of the Art

Chapter Contents

2.1 Introduction

The rapid advancement and deployment of artificial intelligence systems across critical domains, particularly healthcare, have unveiled significant security vulnerabilities that threaten their reliability and trustworthiness. This literature review provides a comprehensive analysis of current threats targeting AI systems and the corresponding defensive strategies developed to mitigate these risks. The growing sophistication of malicious attacks—ranging from adversarial perturbations to data poisoning and model extraction—necessitates a thorough understanding of both offensive techniques and protective mechanisms.

This chapter systematically examines three primary threat categories: adversarial attacks, which exploit model vulnerabilities during inference through carefully crafted input perturbations; data poisoning, which compromises the integrity of training data to manipulate model behavior; and model extraction attacks, which aim to steal intellectual property through strategic querying of deployed models. Each threat type presents unique challenges and requires specialized defense approaches.

Beyond threat characterization, this review analyzes the inherent vulnerabilities of AI systems throughout their lifecycle, from training phase susceptibility to poisoning attacks to inference-time vulnerability against adversarial examples. Furthermore, we survey contemporary defense methodologies, including adversarial training, defensive distillation, anomaly detection techniques, and robustness-enhancing frameworks.

The objective of this state-of-the-art analysis is to establish a solid theoretical foundation for understanding AI security challenges while identifying gaps in existing defense strategies. This comprehensive overview not only informs the experimental components of this work but also contributes to the broader effort of developing more resilient and trustworthy AI systems for healthcare applications and other critical domains.

2.2 Threat Types Against AI Systems

2.2.1 Adversarial Attacks

2.2.1.1 What Are Adversarial Attacks?

Adversarial attacks represent one of the most critical threats to artificial intelligence systems, particularly those based on deep neural networks. These attacks involve the deliberate crafting of inputs that are subtly modified—often in ways that remain invisible to the human eye—yet are sufficient to deceive the model and trigger incorrect predictions. Such manipulations take advantage of the high sensitivity of AI models to small perturbations in input data. Over time, various methods have been developed to generate these attacks, each relying on distinct strategies and tailored to specific constraints. Experiments reported by Papernot et al. (2016)on the MNIST

dataset demonstrate that even minimal modifications to input images can mislead a neural network into making incorrect predictions. These alterations, though typically imperceptible to the human eye, exploit the model's sensitivity to subtle pixel-level variations. In their study, distortions ranged from 0.26% to 13.78%, with an average of 4.06%, and were sufficient to cause misclassification. This highlights the susceptibility of deep learning models to adversarial attacks, even when the perturbations are nearly invisible.

2.2.1.2 Common Methods for Generating Adversarial Examples

To deepen the understanding of adversarial attacks, several well-established techniques are outlined below. Each method demonstrates a specific way of manipulating inputs to mislead deep learning models, highlighting different strategies of optimization, perturbation, or targeted manipulation.

- **FGSM (Fast Gradient Sign Method):** The Fast Gradient Sign Method (FGSM) is one of the simplest and most widely used techniques for generating adversarial examples. Introduced by Ian Goodfellow in 2014 Goodfellow et al. (2014), it gained popularity due to its efficiency and ease of implementation. FGSM leverages the gradients of the model's loss function with respect to the input data to introduce small, directionally guided perturbations. The method is defined by the following formula:

$$\boldsymbol{\eta} = \epsilon \text{sign}\left(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y)\right).$$

Figure 2.1: Equation used in FGSM attack

Where:

  - η: the perturbation added to the input.
  - ϵ: the controlled magnitude of the perturbation.
  - $\nabla_x J(\theta, x, y)$: the gradient of the loss function with respect to the input x.
  - $\text{sign}(\cdot)$: the operator that extracts only the direction (sign) of the gradient for each pixel.

The performance of the FGSM method was evaluated in Goodfellow et al. (2014) across various datasets and architectures. The table below summarizes the error rates induced by FGSM depending on the targeted model and the perturbation parameter ϵ:

Dataset	Model	ϵ	Error caused by FGSM
MNIST	Softmax	0.25	99.9%
MNIST	Maxout Network	0.25	89.4%
CIFAR-10	CNN Maxout	0.10	87.15%

Table 2.1: Error rates caused by FGSM on different datasets and models

- **PGD (Projected Gradient Descent):** The Projected Gradient Descent (PGD) method is an

improved iterative version of FGSM, capable of generating stronger adversarial examples Madry et al. (2017). Unlike FGSM, which applies a single gradient-based update, PGD performs multiple small steps iteratively. After each update, the perturbation is projected back into a permissible region, usually constrained by a norm such as the ℓ_∞ norm.

$$x^{t+1} = \Pi_{x+\mathcal{S}}\left(x^t + \alpha\,\mathrm{sgn}(\nabla_x L(\theta, x, y))\right).$$

Figure 2.2: Equation used in PGD attack

Where:

- x_0: initial point, often defined as the input x plus random noise.
- α: step size.
- Π: projection operator onto the constrained set (e.g., ℓ_∞ ball).
- $\nabla_x L(\theta, x, y)$: gradient of the loss with respect to the input.
- $\mathrm{sign}(\cdot)$: element-wise sign of the gradient (direction per pixel).

The table below illustrates the effectiveness of the PGD attack when applied to a standard, undefended model trained on the CIFAR-10 dataset Madry et al. (2017). A significant drop in accuracy is observed — from over 90% under normal conditions to below 4% under attack — clearly demonstrating the strength of PGD in bypassing non-robust models.

Model Capacity	Natural Accuracy	Accuracy under PGD
Simple	92.7%	0.8%
Wide	95.2%	3.5%

Table 2.2: Natural and adversarial accuracy under PGD for models with different capacities

- **Carlini & Wagner (C&W):** The Carlini & Wagner (C&W) attack is one of the most formidable approaches to fooling machine learning models Carlini and Wagner (2017). Unlike FGSM or PGD, this method solves an optimization problem to identify the minimal perturbation required to induce a misclassification. Although computationally more expensive, its effectiveness is enhanced by the subtlety of the applied modifications, which are often imperceptible. The formulation is as follows: Where:

$$f(x') = \max(\max\{Z(x')_i : i \neq t\} - Z(x')_t, -\kappa).$$

Figure 2.3: Equation used in C&W attack

- $Z(x')$: logits.

- t: target class.
- κ: confidence margin.

In their study, Carlini and Wagner demonstrate that their attack achieves a 100% success rate even against defensively distilled networks, across all three norms (ℓ_0, ℓ_2, ℓ_∞). For instance, on CIFAR-10, the attack succeeds with an average perturbation of just 0.002 under ℓ_∞, and 0.36 under ℓ_2 in the best case. On MNIST, as few as 10 pixels are modified on average under ℓ_0. These results highlight the strength and stealth of the C&W method, capable of bypassing advanced defenses while remaining nearly undetectable.

- **Basic Iterative Method (BIM):** The Basic Iterative Method (BIM) Kurakin et al. (2017) is an iterative version of FGSM that applies multiple small, successive perturbations in the gradient direction. At each step, the image is projected into a constrained neighborhood to remain close to the original. This method enhances attack effectiveness while preserving subtle modifications. The formulation is as follows:

$$\boldsymbol{X}_0^{adv} = \boldsymbol{X}, \quad \boldsymbol{X}_{N+1}^{adv} = Clip_{X,\epsilon}\Big\{\boldsymbol{X}_N^{adv} + \alpha\,\mathrm{sign}(\nabla_X J(\boldsymbol{X}_N^{adv}, y_{true}))\Big\}$$

Figure 2.4: Equation used in BIM attack

Where:

- X: clean image.
- X_N^{adv}: adversarial image after N iterations.
- α: step size per iteration.
- ϵ: maximum allowed perturbation radius around the original image.
- $\mathrm{Clip}_{X,\epsilon}$: projection to remain within the $\ell_\infty(\epsilon)$-ball around X.
- $J(X, y)$: loss function (typically cross-entropy).
- $\nabla_X J$: gradient of the loss with respect to the input.
- $\mathrm{sign}(\cdot)$: gradient sign (pixel-wise).

The table below presents results from applying the BIM attack in a physical-world scenario Kurakin et al. (2017). Adversarial images were printed, photographed, automatically cropped, and then fed into an Inception v3 model. Classification accuracy was measured using Top-1 and Top-5 metrics to evaluate whether BIM-generated perturbations remain effective after physical transformations. The results demonstrate a clear inverse correlation between perturbation magnitude (ϵ) and model accuracy, with higher ϵ values consistently degrading classification performance — both on original digital images and after physical transformation. This empirical evidence confirms BIM's effectiveness in generating adversarial examples that maintain their attack potency in real-world conditions.

ϵ	Digital Images		Physical Images	
	Top-1 Acc.	Top-5 Acc.	Top-1 Acc.	Top-5 Acc.
2	62.6%	86.9%	48.0%	80.4%
4	51.0%	87.3%	41.2%	75.0%
8	28.4%	48.0%	24.5%	31.4%
16	26.5%	49.0%	28.4%	31.4%

Table 2.3: Performance of the BIM method under different perturbation magnitudes (ϵ), evaluated on digital and physical images.

2.2.1.3 Comparative Table of Adversarial Attack Methods:

To summarize the key characteristics of the presented methods, Table compares FGSM, BIM, PGD, and C&W approaches across several criteria:

- **Type:** One-step (single iteration) vs. iterative (multiple adjustments) method.
- **Perturbation Norm:** Constraint on the modification magnitude:
  - ℓ_∞: maximum per-pixel change.
  - ℓ_2: global Euclidean distance.
  - ℓ_0: number of modified pixels.
- **Defense Robustness:** Ability to bypass countermeasures (e.g., adversarial training).
- **Speed/Accuracy:** Trade-off between computational cost and attack success rate.

Criterion	**FGSM**	**BIM**	**PGD**	**C&W**
Type	One-step (single gradient update)	Iterative	Iterative + projection	Advanced optimization
Complexity	Low	Medium	High	Very high
Speed	Fast (single step)	Moderate (multi-step)	Slow (multiple iterations)	Very slow (optimization process)
Defense Robustness	Low (easily detectable)	Moderate (less effective than PGD)	High (bypasses adversarial training)	Very high (evades most defenses)
Perturbation Norm	ℓ_∞ (bounded)	ℓ_∞	ℓ_∞ or ℓ_2	ℓ_0, ℓ_2, ℓ_∞
Limitations	Weak against defenses	Less effective than PGD	Computationally expensive	Extremely computationally intensive

Table 2.4: Comparative Table of Adversarial Attack Methods

This comparative analysis highlights fundamental trade-offs between attack approaches. While fast, simple methods like FGSM demonstrate lower effectiveness against advanced defenses, iterative (PGD) and optimization-based (C&W) techniques achieve greater robustness at higher computational costs.

The optimal method selection thus depends on specific use cases:

- **Rapid testing:** FGSM
- **Robustness evaluation:** PGD
- **Sophisticated attacks:** C&W

Each approach offers distinct advantages tailored to operational priorities – whether prioritizing speed, complexity, or attack efficacy – making methodological choice context-dependent.

2.2.2 Data Poisoning

2.2.2.1 General Definition of Data Poisoning

Data poisoning refers to an attack technique in which an adversary injects manipulated or falsified examples into a model's training data, with the intent of disrupting its behavior. This manipulation can either degrade the overall performance of the system or trigger specific, targeted errors. Such attacks are often difficult to detect, particularly when data collection is uncontrolled or loosely supervised. A clear distinction between a standard deep learning pipeline and one compromised by data poisoning is highlighted in the work of Zhao et al. (2025b). In a typical training process, raw data is collected, preprocessed, and then used to train a model, which is later evaluated on a test set. However, in a poisoned scenario, an adversary injects maliciously crafted samples into the training dataset. These poisoned inputs corrupt the learning process, resulting in a contaminated model that produces incorrect or manipulated predictions during evaluation. This emphasizes the critical impact such attacks can have on model reliability, even when only a portion of the training data is tampered with.

2.2.2.2 Taxonomy of Attacks

The taxonomy of data poisoning attacks can be organized along several dimensions:

- **By Objective:** Poisoning attacks can be categorized according to the attacker's intended goal. This classification primarily distinguishes between targeted and untargeted attacks, depending on the desired impact on the model's predictions.
  - **Targeted Attacks**: In a targeted poisoning attack, the adversary's goal is to cause the model to consistently misclassify a specific input or class. This means that, even if the model performs well overall, it will systematically produce incorrect predictions for particular inputs predefined by the attacker. This category includes attacks such as backdoor attacks and targeted clean-label attacks. In Tolpegin et al. (2020), such an attack is demonstrated in a Federated Learning context using a targeted label-

flipping strategy, where malicious participants locally modify labels from a source class to a target class (e.g., airplane → bird). The results show that with only 10% of participants being malicious, the recall of the targeted class dropped by more than 20%, while the other classes remained nearly unaffected (less than 1%), making the attack both stealthy and effective.

- **Untargeted Attacks**: In an untargeted poisoning attack, the adversary aims to disrupt the overall behavior of the model without focusing on any specific output or class. The goal is to degrade the model's general performance, for instance by increasing the error rate across the entire test set. This category includes methods such as random label flipping, data replacement attacks, and untargeted clean-label poisoning. The Phantom attack Knauer et al. (2024), proposed in the context of semi-supervised learning, involves injecting slightly altered images into the pool of unlabeled data to globally interfere with the training process. With just 5% of poisoned examples, model accuracy dropped by more than 10%. By leveraging public sources like Facebook, this attack remains stealthy, realistic, and hard to detect — highlighting the susceptibility of SSL models to untargeted threats.

- **By Strategy:** Poisoning attacks can also be categorized according to the strategy used. Poisoning strategies refer to how malicious data is crafted—by altering labels (label flipping), modifying the input content while keeping labels unchanged (clean-label), or embedding hidden triggers (backdoor attacks).

  - **Label Flipping Attack**:Label flipping is a form of data poisoning in which the adversary changes the labels of certain training instances without modifying the input data itself. The authors of Shahid et al. (2022) demonstrated the effectiveness of this attack in a human activity recognition (HAR) system based on mobile sensors. Their study showed that flipping just 15% of the labels significantly reduced the model's accuracy, rendering the system almost unusable.

Model	Initial Accuracy	Accuracy with 15% Attack
MLP	94%	39%
Decision Tree	86%	35%
Random Forest	92%	56%
XGBoost	96%	55%

Table 2.5: Impact of 15% adversarial attack on different models

  - **Clean-label Attack:**: Clean-Label Attacks involve injecting malicious examples into the training data while keeping the labels correct, making them particularly difficult to detect. One illustrative example is the Poison Frogs attack Shafahi et al. (2018),

which demonstrates the effectiveness of this approach: by subtly modifying an image from the source class, the authors successfully induce a targeted misclassification of a specific image, without significantly affecting the model's overall performance. The following results summarize the effectiveness of their method in two different learning scenarios:

Scenario	Model Used	# Poisons	Success Rate	Global Accuracy Drop
Transfer Learning	InceptionV3 (ImageNet)	1	100%	$< 0.4\%$
Full Training	AlexNet (CIFAR-10)	50 (watermarked)	$\sim 60\%$ (up to 70%)	Negligible

Table 2.6: Examples of poisoning attack scenarios and their impact on success rate and global accuracy.

- **Backdoor Attack**: A backdoor attack involves embedding hidden behavior into a machine learning model, making it respond to a specific trigger. The goal is for the model to perform normally under typical conditions, but produce an attacker-controlled output when a predefined pattern is present. For example, in a digit recognition model, an attacker might insert a small white square in the corner of certain images of the digit '5' while labeling them as '8'. After training, any image containing this square would be classified as an '8', regardless of its true class. The table below, taken from Khaddaj et al. (2023), shows that some backdoor attacks can manipulate a model with up to 75% accuracy on trigger-injected inputs, while maintaining over 86% accuracy on clean data. This demonstrates both their effectiveness and stealth, even with a low injection rate. DL and CL refer to Dirty-Label and Clean-Label attacks respectively. "CL (no adv.)" represents the non-adversarial variant of the clean-label attack.

Attack Strategy	Poisoning Rate (α)	Clean Accuracy (%)	Backdoor Success Rate (%)
DL	1.5%	86.39	49.57
CL	1.5%	86.89	75.58
CL (no adv.)	5%	86.94	71.68

Table 2.7: Impact of poisoning rate on clean accuracy and backdoor success rate under different attack strategies.

- **By Technical Means:** Poisoning attacks can also be categorized according to the technical methods used to generate malicious data. These include gradient-based optimization, generative models, or stealthy injection into vulnerable data sources.
  - **Gradient-Based Attack:** These attacks generate poisoned examples by leveraging the gradients of the model's loss function in order to indirectly influence the learning process. The goal is to make the model learn a malicious behavior—such as the misclassification of a specific target image—while maintaining the visual integrity of the training data. In the Witches' Brew attack Geiping et al. (2020), the authors craft poisoned samples by aligning their gradients with those of a chosen target. This

forces the optimization process to associate the target image with an incorrect class. The method relies on a gradient alignment function based on angular similarity, and it proves highly effective—even when modifying less than 1% of the training data. The table below summarizes the results achieved:

Dataset	Model	Poisoning Budget	Perturbation (ℓ_∞)	Attack Success Rate (%)
CIFAR-10	ResNet-18	1%	16	90%
ImageNet	ResNet-18	0.1%	8	80%
AutoML (black-box)	–	0.1%	32	Top-5: 100%

Table 2.8: Attack success rates under different datasets, models, poisoning budgets, and perturbation constraints.

The results demonstrate that gradient alignment effectively influences the model's learning process, even with a low poisoning rate. The malicious gradients are sufficient to divert the model's prediction on the target. The high success rate—including on challenging datasets like ImageNet and under black-box settings—confirms that gradient manipulation is a powerful and hard-to-detect strategy.

- **Generative-model-based / Heuristic Attack:** Generative Model-Based Attacks leverage the capabilities of networks such as GANs (Generative Adversarial Networks) to automatically generate malicious training samples. These attacks rely on heuristic methods designed to disrupt the learning process without requiring explicit optimization of the loss function over the entire target model. A notable illustration of this approach is presented by Muñoz-González et al. (2019). in their work on pGAN (poisoning GAN). This model introduces an architecture composed of three modules: a generator, a discriminator, and the target classifier. The goal is to generate poisoned samples that are visually indistinguishable from legitimate data but capable of significantly degrading model performance. The attack is formulated as a minimax game, where the generator aims to fool both the classifier (by increasing its error) and the discriminator (by avoiding detection as an anomaly). A parameter α is introduced to control the trade-off between attack effectiveness and stealth. This approach offers several advantages:
  - It enables large-scale automated poisoning, which is difficult to achieve with traditional bilevel optimization methods.
  - It is more realistic, as the generated data blends more naturally with the real data distribution.
  - It supports both targeted and indiscriminate attacks depending on the attacker's objective.

  However, some limitations remain—most notably the assumption of full knowledge of the target system and the complexity of training GANs effectively.
- **Optimization-based Attacks:** Optimization-based data poisoning attacks frame the

creation of poisoned examples as a bilevel optimization problem. The attacker injects small, correctly labeled (clean label) perturbations into the training set, with the goal of causing a specific target input to be misclassified after training—without degrading the model's overall performance. This approach allows for precise control over the model's behavior. The MetaPoison method Huang et al. (2020) provides a practical solution to this problem by approximating bilevel optimization through meta-learning and truncated unrolling (K = 2) of the training process. Unlike heuristic-based methods such as Feature Collision, MetaPoison is the first clean-label poisoning method effective on models trained from scratch, without requiring knowledge of the final model architecture or weights. It achieves high attack success rates (up to 90%) with only 1% poisoned data, and proves robust across architectures, training settings, and even in real world black-box platforms like Google Cloud AutoML. MetaPoison sets a new standard for general-purpose, optimization-based poisoning attacks.

2.2.3 Model Extraction Attacks

2.2.3.1 Definition of Model Extraction

Zhao et al. (2025a) Model Extraction Attacks (MEA) pose a significant threat to the security of AI systems, as they allow adversaries to retrieve either the exact parameters of a target model M or build a functional approximation of it. A MEA is defined as an attack in which the adversary leverages access to the model's prediction interface (e.g., via queries) to steal, replicate, or approximate its behavior. The attack goals may vary: some aim to reconstruct the exact weights of M, while others seek to train a substitute model M' that replicates the decision boundaries of the original. In Kariyappa and Qureshi (2020), a typical scenario illustrates this process: an adversary sends inputs—such as images of dogs or other data—to the target model, collects the associated predictions, and then uses this labeled dataset to train a clone model. This approach allows the attacker to reproduce the behavior of the original system without needing direct access to its architecture or internal parameters.

2.2.3.2 Black-Box / White-Box Stealing

Model stealing attacks can be broadly categorized based on the attacker's level of access to the target model: **black-box** and **white-box** attacks.

- **Black-Box Model Stealing:**

  In a black-box attack, the adversary has no access to the internal structure or parameters of the target model. Instead, interaction occurs solely through inputs and outputs, often via an API. The attacker aims to reconstruct a functionally equivalent substitute model using input-output pairs. According to Wang et al. (2024), the typical process involves three steps:

1. **Querying the target model:** The attacker sends a large number of inputs—either real or synthetically generated—to the target model.
2. **Collecting the responses:** The predictions or confidence scores returned by the model are recorded.
3. **Training a clone model:** The attacker uses the collected data to train a substitute model that mimics the behavior of the original one.

- **White-Box Model Stealing:**

  In a white-box attack, the adversary has full access to the target model's architecture and internal parameters. This level of access enables more effective replication or manipulation, as the attacker can directly copy the model's structure and weights. Compared to black-box attacks, white-box attacks are generally easier to execute. A typical scenario described in Wang et al. (2024) includes:

  1. **Accessing model details:** The attacker obtains information about the model's architecture and weights.
  2. **Replicating the model:** Using this information, the attacker creates an exact copy of the model by duplicating its structure and parameters.

2.2.3.3 Common Techniques for Model Extraction

- **Query-Based Attacks:** Query-based model extraction attacks involve adversaries attempting to replicate a machine learning model by interacting with it solely through its prediction interface (typically an API), without any access to its internal architecture or parameters. The attacker systematically submits numerous queries—using either real or synthetic inputs—and collects the corresponding outputs, such as predicted labels or confidence scores. These input-output pairs are then used to train a surrogate model that imitates the behavior of the original. This attack vector is particularly threatening in the context of Machine Learning as a Service (MLaaS), where powerful proprietary models are exposed through APIs. Even with this limited interaction, attackers have succeeded in building highly accurate replicas. For instance, research presented in Liang et al. (2024) shows that on certain platforms, such as Microsoft's sentiment analysis API on the Yelp dataset, the stolen models achieved a fidelity score exceeding 90%—meaning their predictions closely mirrored those of the original black-box model. Such findings underscore the critical need to implement robust defenses on publicly accessible ML systems, especially those deployed via commercial APIs.
- **Active Learning:** Active learning is a query-efficient strategy in which the model selectively chooses the most informative examples to be labeled, aiming to minimize the overall number of queries required for training. In the context of model extraction, this approach can significantly enhance attack efficiency. According to findings in Chandrasekaran et al. (2020), the Extended Adaptive Training (EAT) technique—an active learning-based

attack—demonstrated a substantial reduction in the number of queries needed to successfully replicate a target model. Compared to classical Adaptive Retraining, EAT achieved similar or even higher accuracy with far fewer queries. For instance, in the case of the Adult dataset, the number of required queries dropped from over 10,000 to fewer than 50 while maintaining high accuracy. Similar improvements were observed across multiple datasets including Mushroom, Breast Cancer, and Diabetes, illustrating the potential of active learning to accelerate model extraction with minimal interaction.

- **Knowledge Distillation:** Knowledge Distillation (KD) is an effective model extraction technique that enables an attacker to replicate a target model by learning from both its final predictions and the underlying probability distributions. The attacker trains a substitute (student) model to mimic the behavior of the target (teacher) model, enhancing the fidelity of the extraction process. In the study by Ezzeddine et al. (2024) , KD is combined with counterfactual explanations (CFs) — synthetic examples that illustrate minimal changes needed to alter the model's output. The substitute model is trained using a hybrid loss function that includes both classification loss and a distillation loss based on Jensen-Shannon divergence, ensuring closer alignment with the target model's responses. Their experiments demonstrate that the KD-based approach can achieve up to 90% agreement with the target model using just 100 queries, a performance that typically requires ten times more queries with standard supervised training. These findings highlight KD's potential to drastically reduce query overhead while maintaining high extraction accuracy when guided by informative samples like CFs.
- **Data-Free Model Extraction – DFME:** Data-Free Model Extraction is a technique in which an attacker reconstructs or approximates a target machine learning model without access to any real training data. Instead of using natural inputs, the attacker relies on synthetically generated data, often produced by a generative adversarial network (GAN), to query the target model and gather responses. These input-output pairs are then used to train a student model that mimics the target's behavior. Despite the absence of real examples, DFME can achieve surprisingly high accuracy, making it a serious threat, especially when the model is exposed via public APIs. For instance, according to Truong et al. (2021) when applied to the SVHN dataset, the extracted model reached 95.2% accuracy compared to the victim model's 96.2%, requiring about 2 million queries. On CIFAR-10, the student model attained 88.1% accuracy against the original's 95.5%, with a total of 20 million queries. These findings demonstrate the surprising effectiveness of the DFME method even under severely restricted access conditions, underscoring the risks associated with exposing machine learning models through public APIs.

2.3 Vulnerabilities of AI Systems to Malicious Attacks

2.3.1 Poisoning: Vulnerabilities in the Training Process

Main origin: These occur when a model is trained on poorly filtered data or data sourced from unreliable origins.

Exploited weaknesses:

- A malicious actor can insert harmful examples into the training set, disrupting the learning process.
- This manipulation is common in environments where data comes from unverified contributors, such as crowdsourcing.
- The lack of effective mechanisms to detect anomalous data facilitates this type of attack.

2.3.2 Adversarial Examples: Model Sensitivity at Inference

Main origin: These attacks exploit the high sensitivity of AI models to slight, imperceptible alterations in input data.

Exploited weaknesses:

- In certain regions of the input space, models—especially neural networks— exhibit nearly linear behavior, making them vulnerable to small, deliberate perturbations.
- The tendency to overfit training data makes them sensitive to insignificant patterns.
- The lack of integration of adversarial examples during training reduces their ability to handle such threats.

2.3.3 Model Extraction: Intellectual Property Theft Risks

Main origin: Unrestricted exposure of prediction interfaces, such as APIs, facilitates this type of attack.

Exploited weaknesses:

- By submitting a large number of queries and analyzing the responses, an adversary can build a replica of the target model.
- The absence of query volume limits or behavior monitoring enables this kind of copying.
- Some systems reveal too much information, such as confidence scores or gradients making reverse-engineering of the original model easier.

2.4 Defense Methods

2.4.1 Defenses Against Adversarial Examples

2.4.1.1 Adversarial Training

Adversarial training is a simple yet effective defensive approach designed to counter adversarial attacks. The idea is to expose the model to deliberately perturbed inputs during the training phase, in addition to the standard training data. This strategy strengthens the model's ability to withstand adversarial perturbations by training it to recognize and resist such manipulations.

In practice, this involves generating perturbed versions of the input data at each training step typically using methods like FGSM or PGD, and updating the model parameters to ensure strong performance on both clean and adversarial examples.

According to Goodfellow et al. (2014), on the MNIST dataset, a maxout network without adversarial training showed an error rate of 89.4% on adversarial examples. However, after applying FGSM-based adversarial training, the error dropped to 17.9%, without degrading performance on clean data. This demonstrates that incorporating adversarial examples during training significantly improves a model's robustness against targeted perturbations.

2.4.1.2 Defensive Distillation

Defensive distillation is a technique used to protect models against adversarial attacks by training a secondary model (known as the *distilled model*) using the "soft outputs" (probability distributions) generated by a first, previously trained model. The main idea is to make the final model less sensitive to small input perturbations.

In practice, a standard model is first trained on the original dataset. Its probabilistic predictions (rather than hard class labels) are then used to train a second model, with a temperature parameter introduced to smooth the output distributions. This process helps the model learn a more stable representation of the data, enhancing its resistance to gradient-based adversarial manipulations.

The authors of Papernot and McDaniel (2017) demonstrated that an improved version of defensive distillation significantly enhances the robustness of neural networks against attacks like FGSM and JSMA, reducing error rates by over 80% on the MNIST dataset while maintaining a clean data accuracy of 97.28%. Their approach, integrating an "outlier" class and uncertainty estimates, also proves effective against black-box attacks.

However, as noted in Carlini and Wagner (2017), this defense can be bypassed by more advanced attacks optimized under different norms (ℓ_0, ℓ_2, ℓ_∞), which can still fool distilled models with nearly 100% success using often imperceptible perturbations. This highlights that while defensive distillation is effective against simple attacks, it remains insufficient when facing sophisticated adversarial strategies.

2.4.1.3 Random Noise Injection

According to Wang et al. (2024), Random Noise Injection stands out for its simplicity and relative effectiveness. This defense technique involves adding small random perturbations either to the input data or within the internal layers of the model. The goal is to reduce the model's sensitivity to imperceptible changes often exploited by adversarial attacks.

By slightly disrupting the processed information, this approach makes it harder for attacks that rely on precise manipulations to succeed. While it does not offer absolute protection, random noise injection can significantly enhance the model's robustness without substantially degrading its performance on clean (unaltered) inputs.

2.4.1.4 Adversarial Example Detection

Detection of adversarial examples is a defense technique designed to automatically identify manipulated inputs before they are processed by the model. These inputs, which appear visually similar to normal data, are deliberately crafted to mislead machine learning systems.

In this context, the authors of Papernot et al. (2016) proposed a detection method based on pixel regularity. They observed that the sum of squared differences between neighboring pixels is, on average, higher for adversarial examples (2,474) compared to normal images (2,151), suggesting a simple yet limited approach to detecting such attacks.

Similarly, a comparative study conducted by the authors of Aldahdooh et al. (2022) provides an in-depth evaluation of eight adversarial example detection methods across various datasets (MNIST, CIFAR-10, SVHN, Tiny-ImageNet) and different attack types (FGSM, PGD, CW, etc.). While the best detectors achieved over 95% detection accuracy on MNIST, their effectiveness dropped to below 60% on more complex datasets like Tiny-ImageNet.

This study highlights the limited generalization ability of current detection methods and their vulnerability to adaptive attacks, underscoring the urgent need for more robust and transferable defense strategies.

2.4.1.5 Comparative Table of Defenses Against Adversarial Examples

Method	Key Principle	Advantages	Limitations	Effectiveness (Example)
Adversarial Training	Trains the model on adversarial examples generated (e.g., FGSM, PGD, etc.).	• Significant reduction of adversarial errors. • Robustness against known attacks.	• High computational cost. • Risk of overfitting to the attacks used during training.	MNIST: adversarial error $\downarrow$ 89.4% $\rightarrow$ 17.9% (FGSM).
Defensive Distillation	Trains a second model on soft probabilities using a temperature parameter.	• Reduces sensitivity to perturbations. • Effective against FGSM/JSMA attacks.	• Can be bypassed by $\ell_0/\ell_2/\ell_\infty$ attacks. • Increased training complexity.	MNIST: >80% reduction in errors, accuracy maintained at 97.28%.
Random Noise Injection	Adds random noise to inputs or internal layers of the model.	• Simple to implement. • Disrupts gradient-based attacks.	• Limited protection against strong attacks. • Variable impact on clean accuracy.	Relative effectiveness: reduces success of attacks but without full guarantee.
Adversarial Example Detection	Identifies adversarial inputs before they reach the model (e.g., pixel analysis, statistics).	• Prevents attacks at early stage. • High detection rate on simple datasets.	• Poor generalization (e.g., on Tiny-ImageNet). • Vulnerable to adaptive attacks.	MNIST: >95% detection, but <60% on Tiny-ImageNet.

Table 2.9: Comparison of adversarial defense methods.

2.4.2 Defenses Against Data Poisoning

2.4.2.1 Gradient/Representation Clustering (e.g., PCA, k-NN)

This method involves analyzing the model's gradients or internal representations to detect abnormal behaviors. Techniques such as PCA or k-NN can be used to identify suspicious updates or examples, often associated with poisoning attacks, by spotting those that deviate from expected distributions. As shown in Tolpegin et al. (2020), applying Principal Component Analysis (PCA) to participant updates allows clear identification of attackers. Even with only 2% of malicious participants, they form a distinct cluster, enabling the aggregator to isolate them without accessing raw data. This method remains effective up to 20% of attackers and is robust against learning-induced variations (known as "gradient drift"). In the same vein, Shahid et al. (2022) proposed a defense against Label Flipping attacks based on the k-NN algorithm, using a small set of trusted data. During an attack affecting 30% of the dataset, the accuracy of an MLP model improved from 19% to 89% after correction, and XGBoost's accuracy increased from 27% to 88%. This approach demonstrates that simple local filtering based on similarity can effectively mitigate the impact of poisoned data.

2.4.2.2 Robust Training (RONI, TRIM)

Among the defensive strategies against poisoning attacks, TRIM and RONI are two robust techniques particularly suited to federated learning. TRIM works by iteratively removing suspicious updates with the largest residuals. This method has shown only a 6.1% median increase in MSE, and in some cases, it even improves performance (e.g., a 3.47% decrease in MSE on a health dataset with 8% poisoning). It is also highly efficient, requiring only 0.02 seconds per iteration on complex data. On the other hand, RONI (Reject On Negative Impact), which relies on validation, discards updates that degrade model performance. While effective under low attack rates, RONI becomes unstable when poisoning reaches 16–20%, with MSE sometimes doubling. It is also significantly slower than TRIM, taking up to 15 seconds per update. In summary, TRIM outperforms RONI in terms of robustness, consistency, and computational efficiency, making it a more suitable solution for federated learning environments exposed to adversarial threats Orhan et al. (n.d.).

2.4.2.3 Influence Functions

Influence Functions are a technique originating from robust statistics that estimate the impact of individual training points on a model's predictions without requiring full retraining. The goal is to understand a model's behavior by tracing how specific training examples affect its decisions. In Koh and Liang (2017), the authors adapted this technique to modern, complex models such as neural networks, by introducing an efficient approximation based on Hessian-vector products and gradients. Their experiments demonstrated a strong correlation (Pearson's $R \approx 0.86$ to 0.95

depending on the case) between the influence predicted by their method and the results obtained through actual retraining—even for non-convex or non-differentiable models like smoothed hinge loss SVMs. They also revealed that models are vulnerable to adversarial manipulation of training data: by imperceptibly altering a single example, they were able to reverse the model's prediction in 57% of test cases for a dog/fish classification task. This rate rose to 77% with two altered examples and nearly 100% with ten, emphasizing the critical importance of data robustness. These findings highlight both the usefulness of Influence Functions for model debugging and their potential for security risk analysis.

2.4.2.4 Comparative Table of Defenses Against Poisoning Attacks

Method	Key Principle	Advantages	Limitations	Effectiveness (Example)
Gradient Clustering (PCA, kNN)	Analyzes gradients or internal representations to detect abnormal behaviors using PCA or k-NN.	• Detects attackers even with only 2% malicious participants. • Resistant to gradient drift. • Simple implementation (k-NN).	• Limited effectiveness beyond 20% attackers. • Requires a small trusted dataset (for kNN).	• PCA: Successfully isolates attackers with up to 20% malicious participants. • k-NN: Accuracy improved from 19% to 89% (MLP) after correction.
Robust Training (TRIM)	Iteratively removes suspicious updates with high residuals.	• High robustness (median MSE increase of only 6.1%). • Fast (0.02 s/iteration). • May even improve performance.	Performance depends on poisoning rate	Healthcare dataset: 3.47% MSE reduction at 8% poisoning rate.
Robust Training (RONI)	Rejects updates that degrade performance using validation.	Effective at low poisoning rates.	• Unstable beyond 16–20% poisoning (MSE may double) • Slow (15 s per update).	Degraded performance at high poisoning rates.
Influence Functions	Estimates the influence of training points on model predictions (without retraining).	• Accurately identifies influential points (R $\approx 0.86 - 0.95$). • Applicable to complex models.	• High computational cost (Hessian-vector products). • Vulnerable to targeted attacks.	57% prediction reversal with one perturbed example; up to 100% with ten examples.

Table 2.10: Comparison of adversarial defense methods.

2.4.3 Defending Against Model Extraction

2.4.3.1 Behavioral Analysis (Stateful Detection, Suspicious Queries)

Behavioral analysis involves monitoring queries made to a machine learning model in order to detect abnormal usage patterns indicative of attacks, such as model extraction or API abuse. For instance, Kariyappa and Qureshi (2020) demonstrated that attacks like KnockoffNets often generate a high proportion of out-of-distribution (OOD) queries, which can be detected using metrics like the maximum softmax probability. By leveraging this behavior, their system was able to reduce the accuracy of the stolen model from 63.6% to 14.3% on the Flowers-17 dataset, while maintaining a high accuracy (91%) for legitimate users. Similarly, the PRADA approach Juuti et al. (2019) analyzes the distribution of distances between consecutive queries. When this distribution deviates from a normal pattern, an alert is triggered. PRADA successfully detected 100% of extraction attacks on MNIST and GTSRB datasets, with 0% false positives (e.g., using a threshold $\delta = 0.96$ on MNIST). It was also able to identify the PAPERNOT attack after only 120 queries— before the substitute model achieved high performance. These results highlight behavioral analysis as an effective and general-purpose defense mechanism, even without access to the target model's internal details.

2.4.3.2 Model Watermarking (API Watermarking):

Model watermarking, or API watermarking, is a security technique designed to protect the intellectual property of AI models by embedding a subtle signature into their outputs. The goal is to deter model theft through extraction attacks—where an adversary clones a model via repeated API queries—by enabling the original owner to prove ownership of the copied model. The work presented in Szyller et al. (2021) introduces an innovative approach: instead of embedding the watermark during training, DAWN dynamically alters a very small fraction (<0.5%) of API responses by injecting incorrect predictions, creating a trigger set that gets learned by the stolen model. The key results include:

- **Effectiveness:** DAWN withstands two extraction attacks (PRADA and KnockOff) with a detection probability greater than $1 - 2^{-64}$
- **Preserved Accuracy:** The original model's accuracy is barely impacted (only a 0.03–0.5% decrease).
- **Robustness:** The watermark remains intact despite evasion attempts (e.g., regularization, noise injection), and is undetectable by the attacker.

Unlike traditional watermarking methods, DAWN does not alter the training phase and binds the watermark to the malicious client, providing proactive protection against model theft via APIs.

2.4.3.3 Limiting the Number of Queries and Adding Noise

This strategy aims to protect machine learning models from theft by disturbing their outputs while preserving their accuracy. The goal is to make adversarial queries ineffective without impacting legitimate users. The study in Lee et al. (2018), introduces the "Reverse Sigmoid" technique an intelligent noise mechanism that maintains correct labels but distorts the confidence scores (probabilities), As result the stolen model experiences a 20% drop in accuracy or requires 64× more queries to achieve the same performance, whereas the protected model retains its accuracy. This makes it an effective and stealthy defense mechanism.

2.4.3.4 Output Rounding / Perturbation (Prediction Poisoning)

This technique involves deliberately altering a model's predictions such as by rounding probability scores or adding noise to reduce the leakage of sensitive information. The goal is to defend against functionality-stealing attacks while maintaining usability for legitimate users.

According to Orekondy et al. (2019), passive methods like truncation are largely ineffective against model theft. The authors introduce MAD, an active defense that perturbs predictions to disrupt the attacker's training process.as results: The stolen model's accuracy drops by 65%, while the impact on the original model remains minimal (only 1–2%). This makes MAD significantly more effective than traditional approaches.

2.4.3.5 Comparative Table of Defenses Against Model Extraction

Method	Key Principle	Advantages	Limitations	Effectiveness (Example)
Behavioral Analysis	Detection of anomalies in queries (OOD, distances between queries).	• Early detection (e.g., 120 queries for PAPERNOT). • Generic (does not require internal access).	May generate false positives if threshold is poorly calibrated.	• Model accuracy stolen reduced from 63.6% to 14.3% (Flowers-17). • 100% detection on MNIST/GTSRB.
Watermarking	Dynamic insertion of signatures into API predictions.	• Detection > 1-2. • Negligible impact on accuracy (0.03-0.5%) • Undetectable by attacker.	Requires a client tracking mechanism.	• Resistant to PRADA and KnockOff attacks. • Watermark persists despite evasion attempts.
Limitation + Noise (Reverse Sigmoid)	Addition of intelligent noise to perturb probabilities.	• Protected model remains accurate • Forces the attacker to make 64× more queries.	Can be bypassed by sophisticated adaptive attacks.	• Stolen model accuracy decreases by 20%. • High cost for attacker.
Output Perturbation (MAD)	Active modification of predictions to poison adversarial training.	• Drastic reduction of stolen model accuracy (-65%). • Low impact on original model (12%). .	More complex to implement than passive methods.	Outperforms classical truncation methods.

Table 2.11: Comparison of adversarial defense methods.

2.5 Overview Diagram of Attacks and Defense Strategies:

Following the detailed analysis of adversarial threats and corresponding defense mechanisms, we provide a visual summary that organizes these concepts in a structured and intuitive

format, The diagram, split into two parts for clarity, illustrates the main categories of attacks targeting AI systems (Figure 2.5) and the defense techniques developed to counter them (Figure 2.6).

This high-level overview offers a comprehensive perspective on the security landscape of AI models, highlighting the relationships between different types of attacks and the mitigation strategies that have emerged in recent research.

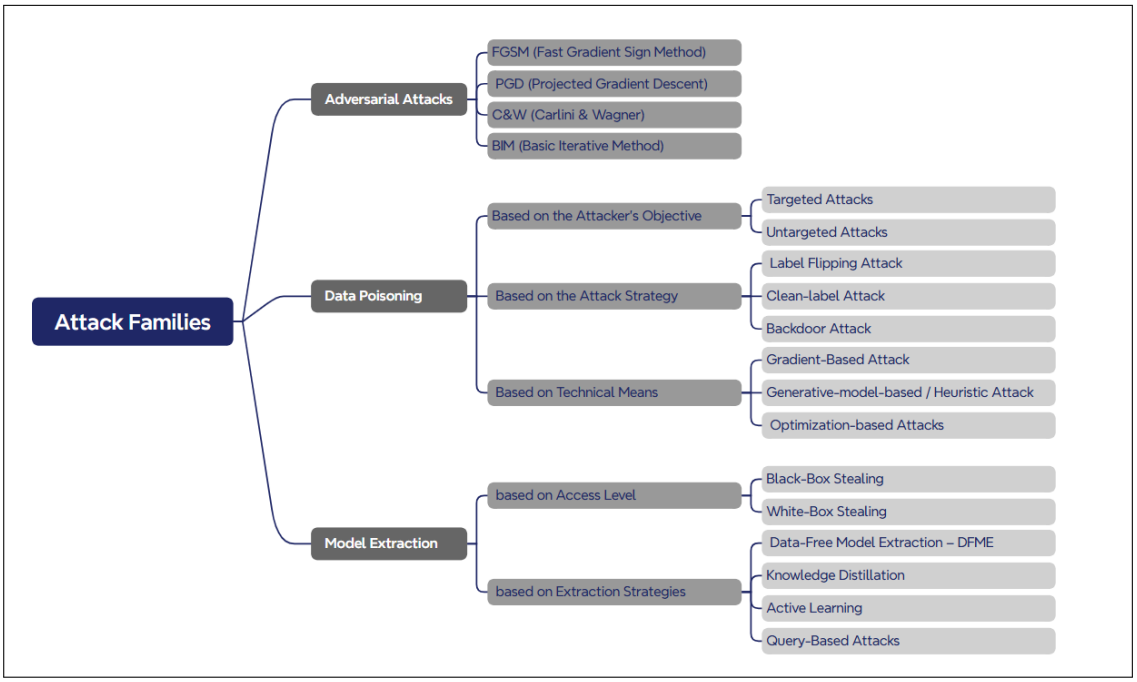


Figure 2.5: Taxonomy of Attack Techniques on AI Models

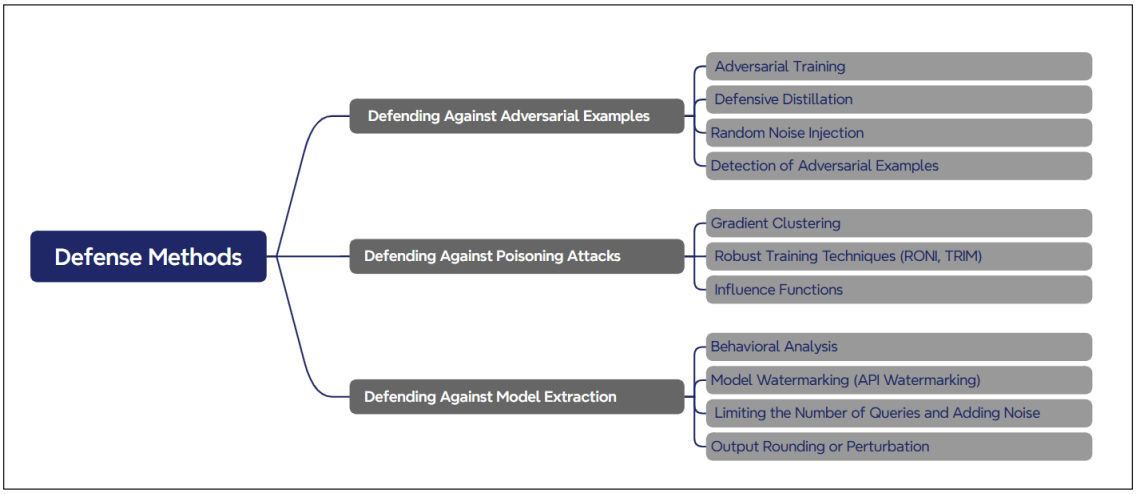


Figure 2.6: Taxonomy of Attack Techniques on AI Models

2.6 Conclusion

This comprehensive literature review has highlighted the extensive and sophisticated threats targeting artificial intelligence systems, particularly in safety-critical domains where security and reliability requirements are paramount. Confronted with the diversity of adversarial attacks, data poisoning techniques, and model extraction attempts, it has become evident that no single solution can provide complete protection. Therefore, a hybrid approach—combining proactive detection mechanisms and robustness enhancement methods—emerges as the most promising strategy. These findings underscore the relevance of our experimental work, which aims to evaluate and propose defensive measures applicable across various domains, offering adaptable protection for both imaging and tabular data applications while strengthening the security and trustworthiness of AI systems in diverse operational contexts.

Chapter 3 Technologies and Tools

Chapter Contents

3.1 Runtime Environment

Google Colab

Google Colab is a cloud-hosted Jupyter notebook service by Google. It runs entirely in the browser with easy Drive integration and optional CPU/GPU/TPU backends. Colab allows seamless collaboration, automatic environment setup, and quick prototyping of machine learning and deep learning projects without local hardware constraints.

Python 3

Python 3 is a general-purpose, readable language with a rich ML ecosystem (NumPy, pandas, PyTorch, etc.). *In this project:* it orchestrates data loading, model training, adversarial attack generation, and performance evaluation. Python's extensive libraries and active community enabled rapid prototyping, efficient numerical computation, and seamless integration with visualization and deep learning tools.

3.2 Data Management & Preprocessing

NumPy

NumPy provides fast n-dimensional arrays and vectorized mathematical operations, forming the backbone of scientific computing in Python. *Use in this work:* it supported tensor↔array conversions, concatenation of adversarial sets, per-feature clipping and dtype control, as well as efficient vectorized metric computations essential for large-scale experiments.

pandas

pandas is a fundamental library for tabular data manipulation, offering powerful DataFrame structures that make column- and row-based operations straightforward. *In this work:* applied to the WDBC dataset for loading CSV files, selecting relevant columns, and mapping diagnostic labels ($B \rightarrow 0$, $M \rightarrow 1$) into numeric values. This preprocessing step ensured the data was properly structured and ready for scaling, enabling consistent and reliable analysis.

scikit-learn

scikit-learn provides a wide range of classical machine learning utilities and preprocessing tools. *In this project:* mainly used for stratified `train_test_split`, ensuring that class proportions were preserved across training and test sets. It was also employed with `StandardScaler`, fit exclusively on the training data before being applied to both training and test sets. This strategy avoided data leakage and ensured a fair evaluation of the MLP models trained on tabular features.

torchvision

torchvision is an extension of PyTorch designed for computer vision tasks, offering standard datasets, pretrained models, and image transformation utilities. *In this work:* applied to the COVID-19 chest X-ray dataset. The `datasets.ImageFolder` function helped organize images into class folders, while a transformation pipeline (`Resize → ToTensor → Normalize` with ImageNet statistics) was used to prepare the training, validation, and test splits. Additionally, a separate adversarial attack pipeline was created without normalization, keeping pixel values in the [0, 1] range for compatibility with ART-based adversarial sample generation.

3.3 Training

PyTorch

PyTorch is a dynamic-graph deep learning framework. In this project it is used to define MLP/CNN models, train with `CrossEntropyLoss` and `AdamW`, stream data via `DataLoader`, and evaluate under `torch.no_grad()`. We run in a CPU ou GPU configuration for reproducibility with fixed seeds . Vision inputs are normalized with ImageNet MEAN/STD consistently at train and test.

3.4 Robustness & Defenses

Adversarial Robustness Toolbox (ART)

ART is an open-source library for adversarial machine learning. It offers a unified interface for major ML frameworks (including PyTorch) and a broad set of standardized adversarial attacks (e.g., FGSM, PGD, BIM, Carlini & Wagner) along with defense utilities and evaluation tools. In this project, ART is used to generate adversarial inputs and to run consistent robustness evaluations on trained models.

3.5 Visualization & Graphics

Matplotlib

Matplotlib was used to produce all figures, including (i) a global pipeline performance bar chart with in-bar percentage labels, (ii) horizontal bar plots of blocking rate by attack type (FGSM, PGD, BIM, CW) at multiple ϵ values, and (iii) robustness curves showing accuracy versus perturbation magnitude for baseline and defence models.

3.6 libraries and imports grouped by family

Family	Imports
Standard library	• `import os` • `import shutil` • `import random` • `import time` • `import copy` • `from zipfile import ZipFile`
NumPy / Pandas	• `import numpy as np` • `import pandas as pd`
scikit-learn	• `from sklearn.model_selection import train_test_split` • `from sklearn.preprocessing import StandardScaler` • `from sklearn.metrics import (accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix, roc_curve, classification_report)`
PyTorch (core)	• `import torch` • `import torch.nn as nn` • `import torch.nn.functional as F` • `import torch.optim as optim` • `import torch.optim.AdamW` • `from torch.utils.data import TensorDataset, DataLoader, Subset, random_split`
Torchvision (CNN)	• `from torchvision import datasets, transforms`
Google Colab (data I/O)	• `from google.colab import drive`
Adversarial Robustness Toolbox (ART)	• `from art.estimators.classification import PyTorchClassifier` • `from art.attacks.evasion import (FastGradientMethod, ProjectedGradientDescent, BasicIterativeMethod, CarliniL2Method)`
Visualization	• `import matplotlib.pyplot as plt`

Table 3.1: Imports grouped by family

Chapter 4 Experimental Methodology

Chapter Contents

4.1 Dataset and Preprocessing

4.1.1 Breast Cancer Wisconsin (WDBC)

Dataset. The *WDBC* dataset contains 569 samples described by 30 numeric features extracted from digitized breast biopsy images (shape/texture descriptors such as *radius, texture, perimeter, area, smoothness, compactness, concavity, concave points, symmetry, fractal dimension*, each provided as `mean`, `se`, and `worst` variants). The `id` column is discarded. The target `diagnosis` is binary (B: benign, M: malignant) and is encoded as `B`→0 and `M`→1.

Provenance. We use the *Wisconsin Diagnostic Breast Cancer (WDBC)* dataset from the *UCI Machine Learning Repository – Health Data* Wolberg and Street (1993).

Preprocessing.

1. **Loading and structuring.** Read `wdbc.data` with the expected column schema; drop `id`.
2. **Separate X/y.**

   $X \in \mathbb{R}^{n\times 30}$ (after removing `id` and `diagnosis`), $y \in \{0,1\}^n$ via $\{\texttt{B} \mapsto 0,\ \texttt{M} \mapsto 1\}$.
3. **Stratified train/test split.** 70% training / 30% test with class-proportion preservation (`stratify=y`).
4. **Standardization (z-score).** Fit a `StandardScaler` **only on X_{train}**, then apply it to X_{train} and X_{test}:

   $$z = \frac{x - \mu_{\text{train}}}{\sigma_{\text{train}}}.$$

   This puts each feature on a comparable scale (mean $\approx$ 0, spread $\approx$ 1) and prevents information leakage from test to train. *Important (attacks): the adversarial budgets ϵ are defined in this standardized space.*
5. **Preparation for modeling.** Convert to PyTorch tensors (`float32` for X, `long` for y), then build `TensorDataset` and `DataLoader` with batch size 64 (`shuffle=True` for training, `False` for test).

Seed. `SEED=42` for the split (scikit-learn `random_state=42`); NumPy/PyTorch seeds set for reproducibility.

Class distribution. In the canonical WDBC split, benign vs. malignant counts are: `B`=**357**, `M`=**212**. With a 70/30 stratified split, this yields **398** training and **171** test samples:

	Train (70%)	Test (30%)
Benign (B=0)	250	107
Malignant (M=1)	148	64
Total	398	171

Table 4.1: Class distribution after stratified split

Data quality. All features are numeric; in our copy of WDBC, no missing values were found. The identifier `id` is dropped prior to modeling.

4.1.2 COVID-19 Radiography

Dataset. We use the binary subset {`COVID`, `Normal`} of the *COVID-19 Radiography Database*. Only these two classes are retained; other categories present in some releases (e.g., `Viral Pneumonia`, `Lung Opacity`) are excluded. Images are reorganized into an `ImageFolder`-compatible layout with one directory per class; non-image files are ignored via filename extension filtering (`.png`/`.jpg`/`.jpeg`/`.bmp`). All images are resized to 224×224 during preprocessing.

Provenance. The data originate from a public repository (*COVID-19 Radiography Database*) Chowdhury et al. (2020b); associated publications as recommended by the authors Chowdhury et al. (2020a); Rahman et al. (2021)

Preprocessing.

1. **Reorganization (binary subset).** Copy images for `COVID` and `Normal` into a clean two-folder structure (`covid_data_prepared/COVID` and `covid_data_prepared/Normal`), keeping only valid image files.
2. **Balanced per-class subsampling.** To control runtime and class balance, we cap each class at a maximum of `PER_CLASS` images (e.g., 500) after random shuffling; if a class has fewer images, we keep all available.
3. **Stratified split on the reduced set.** For each class independently, we compute:

$$n_{\text{test}} = \text{round}(0.20\, n), \qquad n_{\text{val}} = \text{round}\big(0.10\, (n - n_{\text{test}})\big),$$

   and allocate the remaining images to training. This preserves class proportions in `train`, `val`, and `test`. (Example with `PER_CLASS`= 500: per class $\Rightarrow$ train 360, val 40, test 100; totals $\Rightarrow$ train 720, val 80, test 200.)
4. **Transforms.**
   - *Training:* `Resize(224)`, `RandomHorizontalFlip`, `ToTensor`, `Normalize` with per-channel ImageNet stats ($\mu = [0.485, 0.456, 0.406]$, $\sigma = [0.229, 0.224, 0.225]$).
   - *Validation/Test:* `Resize(224)`, `ToTensor`, `Normalize` (same stats).
   - *Attack (generation):* `Resize(224)`, `ToTensor` only (i.e., images in $[0, 1]$).

Important (attacks): adversarial examples are generated on inputs in $[0, 1]$; the model's normalization is applied in the classifier/wrapper so that attacks are performed in image space while inference remains consistent.

5. **Preparation for modeling.** Build `ImageFolder` datasets with the above transforms; create `Subsets` using the class-wise indices for `train`, `val`, `test`, and an `attack_test` subset (same images as test, but without normalization for attack generation). Use `DataLoaders` with batch size 32 for training and 128 for evaluation/attack; enable shuffling for training only.

Seed. We fix `SEED=42` for shuffling, per-class subsampling and class-wise splits (rng = NumPy 42); PyTorch and Python seeds are also set for reproducibility.

Summary (parameters).

Element	Setting
Classes kept	`COVID`, `Normal` (binary subset)
Balancing	Cap at `PER_CLASS` images per class (e.g., 500).
Split	Test = 20%; Validation = 10% of remaining; rest = Train; stratified per class.
Image size	224×224
Train augmentation	Random horizontal flip
Normalization	Per-channel ImageNet stats on train/val/test
Attack data scale	$[0, 1]$ (no normalize); normalization handled in wrapper
Batch sizes	Train = 32; Eval/Attack = 128
Seed	42 (reproducibility)

Table 4.2: Summary of preprocessing parameters for COVID-19 Radiography .

4.2 Model Definitions

4.2.1 Multilayer Perceptron (MLP) — Architecture for WDBC

Objective. For binary classification on the Breast Cancer Wisconsin (Diagnostic) dataset, we use a multilayer perceptron (MLP) tailored to standardized tabular features and capable of capturing non-linear relationships.

Architecture. The network consists of four fully connected layers arranged in a funnel:

- **Input:** 30 features.
- **Hidden 1:** 128 units → `BatchNorm1d` → ReLU → Dropout ($p = 0.5$).
- **Hidden 2:** 64 units → `BatchNorm1d` → ReLU → Dropout ($p = 0.5$).
- **Hidden 3:** 32 units → `BatchNorm1d` → ReLU → Dropout ($p = 0.5$).
- **Output:** 2 units (logits for benign/malignant).

Forward behavior. The 30 z-score standardized features pass through, for each hidden block, a sequence of Linear → `BatchNorm1d` → ReLU → Dropout. The final linear layer outputs raw *logits*; no softmax is applied in the forward pass since `CrossEntropyLoss` operates directly on logits during training.

Design rationale.

- **BatchNorm1d** stabilizes activation scales and eases optimization.
- **ReLU** provides effective non-linearity.
- **Dropout** ($p = 0.5$) delivers strong regularization for a relatively small dataset.
- **Widths 128 → 64 → 32** balance capacity and compactness via progressive dimensionality reduction.

A compact MLP with `BatchNorm1d` + ReLU + Dropout after each hidden layer, producing two logits from 30 standardized features, well matched to WDBC.

4.2.2 Convolutional Neural Network (CNN) — Architecture for COVID-19 Radiography

Objective. A compact convolutional network for binary classification of chest X-ray images (`COVID` vs. `Normal`).

Architecture. The model stacks three convolutional blocks followed by global pooling and a linear classifier:

- **Block 1:** `Conv2d(3→32, 3×3, padding=1, bias=False)` → `BatchNorm2d(32)` → ReLU → `MaxPool2d(2)`.
- **Block 2:** `Conv2d(32→64, 3×3, padding=1, bias=False)` → `BatchNorm2d(64)` → ReLU → `MaxPool2d(2)`.
- **Block 3:** `Conv2d(64→128, 3×3, padding=1, bias=False)` → `BatchNorm2d(128)` → ReLU → `MaxPool2d(2)`.
- **Head:** `AdaptiveAvgPool2d(1×1)` (global average pooling) → flatten → `Dropout(`$p = 0.3$`)` → `Linear(128→2)` (logits).

Forward behavior. An input tensor of shape (batch, 3, 224, 224) passes through the three conv blocks; spatial resolution is downsampled by a factor of $2 \times 2 \times 2$. Global average pooling yields a (batch, 128, 1, 1) tensor, which is flattened to (batch, 128), passed through Dropout, then a final linear layer outputs two raw *logits*. No softmax is applied in the forward pass (logits are suitable for `CrossEntropyLoss`).

Feature extraction. An auxiliary method `extract_features` returns the 128-dimensional pooled embedding (before Dropout/Linear), which can be reused for downstream tasks (e.g., adversarial example detection).

Initialization. Convolutional weights are initialized with Kaiming-normal (adapted to ReLU), batch-normalization scales are set to 1 and biases to 0, while the final linear layer is initialized using Xavier-uniform with zero bias.

Design rationale. The use of 3×3 convolutions with padding preserves local spatial detail. Batch normalization stabilizes and accelerates training. MaxPooling progressively reduces spatial dimensions and computation cost. Global average pooling followed by a lightweight classifier reduces the number of trainable parameters and mitigates overfitting. Finally, Dropout($p = 0.3$) provides additional regularization while maintaining model capacity.

4.3 Baseline (Training and Evaluation)

4.3.1 Multilayer Perceptron (MLP)

Setup. The baseline MLP is trained on the WDBC tabular dataset (30 standardized features; binary classification, *positive class* = 1). Standardization uses a `StandardScaler` fit only on the training split, then applied to train/test (stratified splits).

Model. A compact feed-forward network over the 30-D vector (linear layers with ReLU and dropout), ending in a 2-logit classifier.

Training (baseline).

- **Loss:** `CrossEntropyLoss`
- **Optimizer:** Adam (lr = 10^{-3}, weight decay = 10^{-4})
- **Epochs:** 30
- **Mini-batching:** PyTorch `DataLoader` on the training split

Each iteration follows the standard loop: forward → CE loss → `optimizer.zero_grad()` → `backward()` → `step()`. Mean training loss is logged per epoch.

Clean evaluation. With `model.eval()`, we evaluate on the held-out test set and report Accuracy, Precision, Recall, F1, and ROC–AUC (from the positive-class probability `softmax(logits)[:, 1]`).

Adversarial evaluation. The trained model is wrapped with ART's `PyTorchClassifier` to preserve preprocessing and per-feature `clip_values` in the standardized space. Robustness is assessed using standard attacks (FGSM, PGD, BIM, C&W-L2).

Baseline MLP pipeline

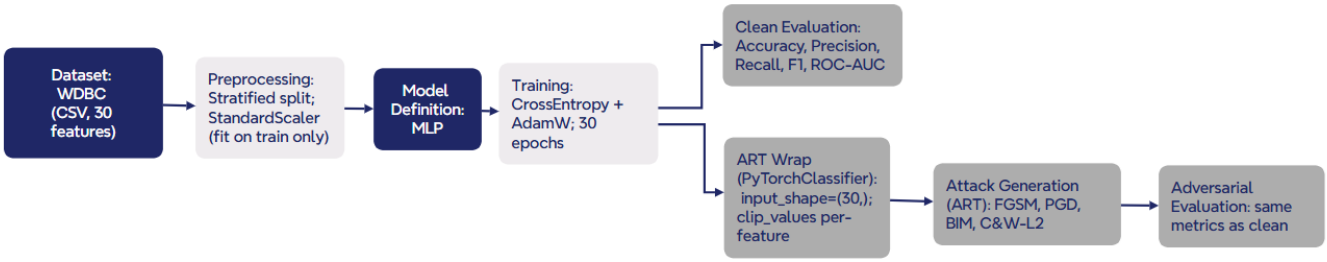


Figure 4.1: Baseline MLP pipeline

4.3.2 Convolutional Neural Network (CNN)

Setup. The baseline CNN is trained on the COVID-19 chest X-ray dataset with stratified train/val/test splits. Images are resized to 224×224 and normalized with ImageNet statistics (MEAN $= [0.485, 0.456, 0.406]$, STD $= [0.229, 0.224, 0.225]$). The *positive class* is COVID. A separate "attack" view (without normalization) keeps inputs in $[0, 1]$ for ART.

Model. `SimpleCNN`: three Conv–BN–ReLU–MaxPool blocks (channels 32/64/128), Global Average Pooling, Dropout(0.3), and a 2-class linear head (logits).

Training (baseline).

- **Loss:** Cross-Entropy
- **Optimizer:** AdamW (lr $= 10^{-3}$, weight decay $= 10^{-4}$)
- **Epochs / Early stopping:** up to 30 epochs; best checkpoint by validation loss with patience $= 5$
- **Batches:** 32 (train), 128 (val/test); **Device:** CPU with fixed seeds

Clean evaluation. On the normalized test set, we report **Accuracy**, **Precision/Recall/F1** with `pos_label` = COVID, and **ROC–AUC** from the softmax probability of the COVID class.

Adversarial evaluation. The trained CNN is wrapped with ART's `PyTorchClassifier`, using `clip_values` $= (0, 1)$ and `preprocessing` $=$ (MEAN, STD) so attack inputs match training normalization. Robustness is measured under standard attacks (FGSM, PGD, BIM, C&W-L2).

Baseline CNN pipeline

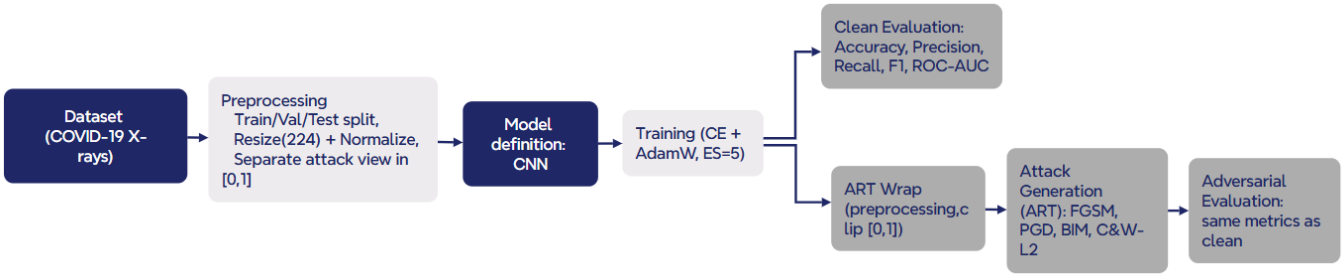


Figure 4.2: Baseline CNN pipeline

4.4 Adversarial Attacks (Methodological Details)

4.4.1 Common setup.

All adversarial examples are generated with the Adversarial Robustness Toolbox (ART 1.17.1). We attack the final trained model and evaluate on the original test split. For each attack, we report Accuracy, Precision, Recall, F1, and ROC–AUC computed on the same items as in the clean evaluation. Generation is streamed in mini-batches and each perturbed batch is immediately evaluated, ensuring stable memory usage and reproducibility.

- **Untargeted** setting for all attacks.
- **Clipping / validity constraints:** ART's `clip_values` keeps perturbed inputs within the valid domain of each modality.
- **Metric consistency:** identical metrics and positive class convention as in the clean evaluation (WDBC: malignant; COVID-19: COVID).

4.4.2 Tabular setting (Breast Cancer Wisconsin, MLP)

Features are standardized with a z-score (`StandardScaler` fit only on train). Attacks therefore operate in the standardized space; the ART classifier uses per-feature `clip_values` derived from the training distribution (min/max after scaling).

FGSM (ℓ_∞). Grid: $\varepsilon \in \{0.1, 0.2, 0.3\}$.

Implementation: `FastGradientMethod(estimator=classifier, eps=`ε`)`.

PGD (ℓ_∞). Grid: $\varepsilon \in \{0.1, 0.2, 0.3\}$; step size $\varepsilon_{\text{step}} = \varepsilon/10$; 20 iterations; no random init (diagnostic run).

Implementation: `ProjectedGradientDescent(..., eps=`ε`, eps_step=`$\varepsilon/10$`, max_iter=20, norm=`∞`, num_random_init=0)`.

BIM (ℓ_∞). Grid: $\varepsilon \in \{0.1, 0.2, 0.3\}$; step size $\varepsilon_{\text{step}} = \varepsilon/10$; 10 iterations.

Implementation: `BasicIterativeMethod(..., eps=`ε`, eps_step=`$\varepsilon/10$`, max_iter=10)`.

Carlini & Wagner L2 (C&W-L2). Two configurations:

- **Fast:** `confidence=0`, `max_iter=75`, `binary_search_steps=1`, `learning_rate=0.02`, `initial_const=0.3`, `batch_size=64`.
- **Strong:** `confidence=0`, `max_iter=500`, `binary_search_steps=7`, `learning_rate=0.01`, `initial_const=0.01`, `batch_size=64`.

Attack Summary

Attack	Norme	ε	Step α	Iters	Init	Notes
FGSM	ℓ_∞	$\{0.1, 0.2, 0.3\}$	–	1	–	per-feature `clip_values`
PGD	ℓ_∞	$\{0.1, 0.2, 0.3\}$	$\varepsilon/10$	20	none	norm=∞
BIM	ℓ_∞	$\{0.1, 0.2, 0.3\}$	$\varepsilon/10$	10	–	I-FGSM
C&W-L2 (fast)	ℓ_2	–	–	75	–	conf=0, bs=64, lr=0.02, c0=0.3, bsearch=1
C&W-L2 (strong)	ℓ_2	–	–	500	–	conf=0, bs=64, lr=0.01, c0=0.01, bsearch=7

Table 4.3: MLP — Attack Hyperparameters (WDBC)

4.4.3 Imaging setting (COVID-19 Radiography, SimpleCNN)

Images are resized to 224×224 and normalized with training statistics during model training. For attacks, we expose a $[0, 1]$ view of inputs to ART and apply the same normalization via the classifier wrapper's `preprocessing=(mean, std)`. Pixel values are bounded with `clip_values=(0,1)`.

ART classifier (wrapper). `PyTorchClassifier(model, loss=CE, optimizer=Adam(1e-3), input_shape=(3,224,224), nb_classes=2, clip_values=(0,1), preprocessing=(mean,std))`.

FGSM (ℓ_∞). Grid: $\varepsilon \in \{2/255, 4/255, 8/255\}$.

Implementation: `FastGradientMethod(estimator=clf, eps=`ε`, batch_size=BATCH_EVAL)`.

PGD (ℓ_∞). Grid: $\varepsilon \in \{2/255, 4/255, 8/255\}$; step size $\varepsilon_{\text{step}} = \varepsilon/8$; 20 iterations; no random init.

Implementation: `ProjectedGradientDescent(..., eps=`ε`, eps_step=`$\varepsilon/8$`, max_iter=20, num_random_init=0)`.

BIM (ℓ_∞). Grid: $\varepsilon \in \{2/255, 4/255, 8/255\}$; step size $\varepsilon_{\text{step}} = \varepsilon/12$; 12 iterations.

Implementation: `BasicIterativeMethod(..., eps=`ε`, eps_step=`$\varepsilon/12$`, max_iter=12)`.

Carlini & Wagner L2 (C&W-L2). Untargeted; `confidence=0`, `max_iter=75`, `binary_search_steps=1`, `learning_rate=0.02`, `initial_const=0.3`, `batch_size=32`.

Implemented with `CarliniL2Method(classifier=clf, ...)` and evaluated in streaming.

Attack	Norme	ε (grid)	α (step)	Iters	Init	Notes
FGSM	ℓ_∞	{2/255, 4/255, 8/255}	–	1	–	`clip_values`=(0,1)
PGD	ℓ_∞	{2/255, 4/255, 8/255}	$\varepsilon/8$	20	none	`preprocessing`=(mean,std), untargeted
BIM	ℓ_∞	{2/255, 4/255, 8/255}	$\varepsilon/12$	12	–	I-FGSM
C&W-L2	ℓ_2	–	–	75	–	conf=0, bs=32, lr=0.02, c0=0.3, bsearch=1

Table 4.4: CNN — Attack Hyperparameters (COVID-19)

Attack Summary

4.4.4 Brief theoretical reminder

Let x be an input, y its label, $J(\theta, x, y)$ the training loss, and $\Pi_{\mathcal{S}}$ the projection onto the admissible set.

- **FGSM (ℓ_∞, 1-step):**

$$x_{\text{adv}} = \text{clip}\big(x + \varepsilon \cdot \text{sign}(\nabla_x J(\theta, x, y))\big), \quad \|x_{\text{adv}} - x\|_\infty \le \varepsilon.$$

- **PGD (ℓ_∞, multi-step):**

$$x^{t+1} = \Pi_{B_\infty(x,\varepsilon)}\left(x^t + \alpha \ \text{sign}(\nabla_x J(\theta, x^t, y))\right), \quad t = 0, \dots, T-1.$$

- **BIM (I-FGSM):** same update as PGD, typically without random starts and with a fixed small number of steps.
- **C&W-L2:** minimizes $\|x_{\text{adv}} - x\|_2^2 + c \cdot f(x_{\text{adv}}, y)$ via gradient-based optimization with a binary search on c; `confidence` controls the required margin.

4.4.5 Why these grids?

For standardized tabular features, $\varepsilon \in \{0.1, 0.2, 0.3\}$ explores mild-to-moderate per-feature perturbations (z-score units). For images, $\{2/255, 4/255, 8/255\}$ are conventional pixel-space budgets covering low, medium, and strong ℓ_∞ noise. Step sizes ($\varepsilon/8$–$\varepsilon/12$) keep iterative methods stable and comparable; the "no random init" PGD runs isolate the effect of ε (stronger PGD with random restarts can be added as an ablation).

4.5 Defenses (Training/Usage Protocols)

4.5.1 Adversarial Training (Mixed Adversarial Training, MAT)

Definition & principle. Mixed adversarial training teaches the model to handle small "stress tests." For each mini-batch, we keep some examples clean and create slightly perturbed versions of others. We then learn from both at the same time, so the model becomes reliable not only on clean data but also under small, plausible changes.

More formally, MAT exposes each mini-batch to a blend of clean $(1 - \rho)$ and adversarial ρ samples generated on-the-fly with short gradient-based attacks (FGSM/PGD/BIM). The total

loss combines both parts with a weight $\lambda \in [0, 1]$:

$$\mathcal{L}_{\text{MAT}}(\theta) = (1-\lambda)\,\mathcal{L}\big(f_\theta(x), y\big) \;+\; \lambda\,\mathcal{L}\big(f_\theta(x^{\text{adv}}), y\big), \qquad \|x^{\text{adv}} - x\|_\infty \le \varepsilon,$$

where x^{adv} is crafted online under an ℓ_∞ budget ε and then clamped back to the valid domain of the current feature space (per-feature bounds in standardized tabular data, or normalized bounds for images).The overall workflow is illustrated in Figure below, which shows how each mini-batch is split into clean $(1-\rho)$ and perturbed ρ parts, their losses are weighted by $(1-\lambda)$ and λ, and the model is updated accordingly.

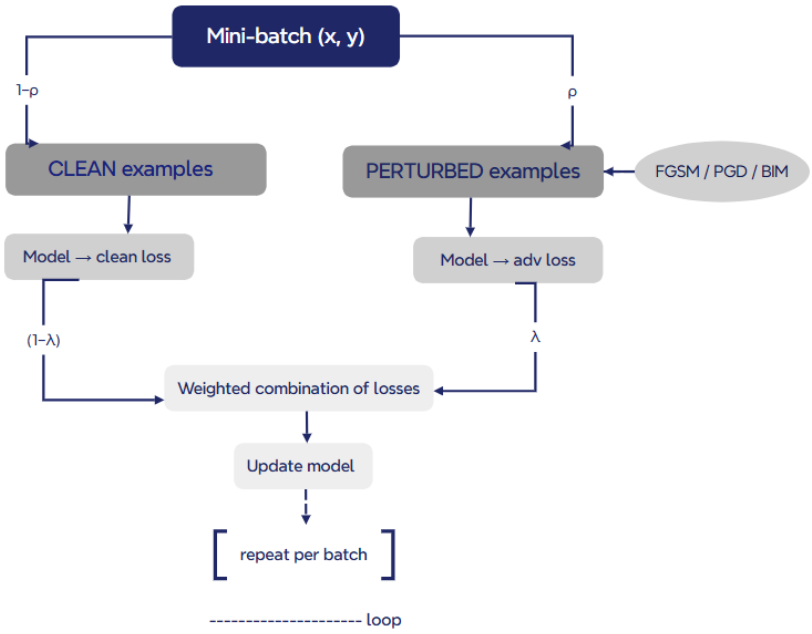


Figure 4.3: Mixed adversarial training pipeline

Usage protocol (shared)

- **Partial batch mix:** attack only a fraction ρ of every mini-batch to control cost.
- **Attack rotation:** alternate FGSM → PGD → BIM batch-by-batch to avoid overfitting to one recipe.
- **Optimization:** AdamW (lr = 10^{-3}, weight decay = 10^{-4}), ~30 epochs, early stopping on validation loss (patience ≈ 5).
- **Clamping:** after each adversarial step, clamp inputs to the valid domain of the current feature space.

Scenario A—Tabular (WDBC, MLP)

- **Preprocessing:** z-score standardization (`StandardScaler` fitted on train only). Attacks operate in this standardized space.
- **Per-feature safety:** after each step, clamp by train-set per-feature min/max.
- **Hyperparameters (MAT-MLP):**

- **Mixing:** $\lambda = 0.50$, $\rho = 0.50$, training budget $\varepsilon_{\text{train}} = 0.2$ (standardized space).
- **Attacks:** FGSM (1 step) → PGD (5 steps, $\alpha = \varepsilon/5$, random start) → BIM (10 steps, $\alpha = \varepsilon/10$).
- **Optimizer:** AdamW (lr = 10^{-3}, wd = 10^{-4}); early stopping.

Scenario B—Imaging (COVID-19 Radiography, CNN)

- **Preprocessing:** inputs normalized with `Normalize`$(\boldsymbol{\mu}, \boldsymbol{\sigma})$. Adversarial examples are generated in the normalized space (not raw pixels).
- **Consistent steps:** re-weight sign updates by $1/\boldsymbol{\sigma}$ (effective step $\alpha/\boldsymbol{\sigma}$) to stay consistent with normalization.
- **Normalized clamping:** after each step, clamp to $\left[(0-\boldsymbol{\mu})/\boldsymbol{\sigma},\ (1-\boldsymbol{\mu})/\boldsymbol{\sigma}\right]$, corresponding to pixel range $[0, 1]$.
- **Hyperparameters (MAT-CNN):**
  - **Mixing:** $\lambda = 0.30$, $\rho = 0.50$, training budget $\varepsilon_{\text{train}} = 4/255$.
  - **Attacks:** FGSM (1) → PGD (3, $\alpha = \varepsilon/4$, random start) → BIM (5, $\alpha = \varepsilon/10$).
  - **Optimizer:** AdamW (lr = 10^{-3}, wd = 10^{-4}); early stopping.

4.5.2 Defensive Distillation

Definition & principle. Defensive distillation trains a smaller (or equal) *student* to imitate a stronger *teacher*. Beyond hard labels, the student also learns from the teacher's *soft class probabilities* at temperature T, which encode class similarity and smooth the decision boundary. More formally, we first train a teacher with standard cross-entropy, then *freeze* it and train the student with a weighted combination of a KL term at temperature T and the usual hard-label loss:

$$\mathcal{L}_{\text{distill}} = \alpha\, T^2\, \mathrm{KL}\big(p_t^{(T)} \,\|\, p_s^{(T)}\big) + (1-\alpha)\, \mathrm{CE}(z_s, y), \qquad p^{(T)} = \mathrm{softmax}(z/T),$$

where z_t and z_s are teacher and student logits, y are ground-truth labels, T is the temperature, and $\alpha \in [0, 1]$ mixes the two terms. The teacher is kept in `eval()` with gradients disabled (no updates); at inference, we deploy the student only. The full distillation workflow is summarized in Figure 4.4, showing Phase 1 (train the teacher), Phase 2 (produce softened targets at temperature T), and Phase 3 (train the student with the hybrid loss and deploy it at inference).

Figure 4.4: Defensive distillation pipeline

Usage protocol (shared)

- **Teacher:** cross-entropy; AdamW (lr = 10^{-3}, weight decay = 10^{-4}); early stopping (patience = 5).
- **Student (distillation):** temperature $T = 8.0$, mix $\alpha = 0.7$; loss $\alpha T^2\,\mathrm{KL}(p_t^{(T)} \| p_s^{(T)}) + (1-\alpha)\,\mathrm{CE}(z_s, y)$; AdamW (lr = 10^{-3}, wd = 10^{-4}); early stopping (patience = 5).
- **Teacher frozen:** `eval()` mode, gradients disabled (no updates) while training the student.

Scenario A—Tabular (WDBC, MLP)

- **Preprocessing:** z-score standardization (`StandardScaler` fitted on train only); attacks (for evaluation) operate in the standardized space.
- **Architecture:** teacher = MLP; student = same MLP (or a smaller variant).
- **Hyperparameters:** $T = 8.0$, $\alpha = 0.7$; AdamW (lr = 10^{-3}, wd = 10^{-4}); early stopping (patience = 5).

Scenario B—Imaging (COVID-19 Radiography, CNN)

- **Preprocessing:** inputs normalized with `Normalize`$(\boldsymbol{\mu}, \boldsymbol{\sigma})$; adversarial evaluations (reported elsewhere) are conducted in the normalized space.
- **Architecture:** teacher = CNN; student = same backbone (or a compact variant).
- **Hyperparameters:** $T = 8.0$, $\alpha = 0.7$; AdamW (lr = 10^{-3}, wd = 10^{-4}); early stopping (patience = 5).

4.5.3 Random Noise Injection (RNI)

Definition & principle. Random Noise Injection teaches the model to stay stable under small, natural "jitter." For each mini-batch, we keep a clean copy of the inputs and create a slightly noisy copy; learning from both encourages predictions that do not flip under minor perturbations. More formally, RNI augments each mini-batch by adding Gaussian noise and training on clean and noisy versions together. For a batch (x, y), form

$$\tilde{x} = \text{clamp}(x + \eta), \qquad \eta \sim \mathcal{N}(0, \sigma^2 I),$$

and optimize

$$\mathcal{L}_{\text{RNI}} = (1 - \lambda_{\text{noise}})\, \text{CE}\big(f_\theta(x), y\big) \;+\; \lambda_{\text{noise}}\, \text{CE}\big(f_\theta(\tilde{x}), y\big),$$

with noisy inputs clamped back to the valid domain of the current feature space (per-feature bounds in standardized tabular data; normalized bounds for images). The overall loop is illustrated in Figure below, which shows how each batch is duplicated into clean and noisy copies, their losses are weighted by $(1 - \lambda_{\text{noise}})$ and λ_{noise}, and the model is updated accordingly.

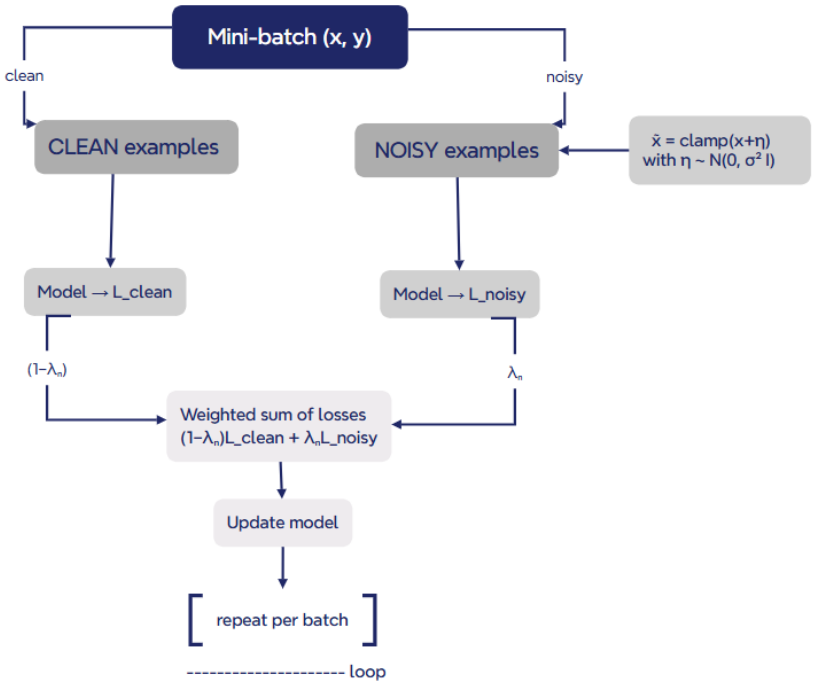


Figure 4.5: Random Noise Injection pipeline

Usage protocol (shared)

- **Mixing weight:** $\lambda_{\text{noise}} = 0.30$ (used across models).
- **Optimization:** AdamW (lr $= 10^{-3}$, weight decay $= 10^{-4}$), ~30 epochs, early stopping on validation loss (patience ≈ 5).
- **Clamping:** after adding noise, clamp inputs back to the valid domain of the current feature space.

Scenario A—Tabular (WDBC, MLP)

- **Noise & space:** add Gaussian noise in the standardized (z-score) space with standard deviation $\sigma = 0.15$.
- **Per-feature clamp:** after corruption, clamp by train-set per-feature min/max in the standardized space.
- **Objective:** same weighted sum as in Definition & principle with $\lambda_{\text{noise}} = 0.30$.

Scenario B—Imaging (COVID–19 Radiography, CNN)

- **Noise & space:** draw $z \sim \mathcal{N}(0, \sigma_{\text{img}}^2 I)$ in the *pixel* domain with $\sigma_{\text{img}} = 4/255$; add in the *normalized* space as

$$x \leftarrow x + \frac{z}{\boldsymbol{\sigma}} \quad .$$

- **Normalized clamping:** after addition, clamp to $\left[(0-\boldsymbol{\mu})/\boldsymbol{\sigma},\ (1-\boldsymbol{\mu})/\boldsymbol{\sigma}\right]$, corresponding to pixel range $[0, 1]$.

- **Objective:** same weighted sum as in Definition & principle with $\lambda_{\text{noise}} = 0.30$.

4.5.4 Detection of adversarial Examples

Definition & principle. An *adversarial example* is an input that has been slightly perturbed so that the model makes an error. Detection inserts a *pre-classifier filter* that decides whether an input is clean or adversarial, with the option to reject it before any clinical decision is made. The objective is to maximize the adversarial true positive rate (TPR) while keeping the false positive rate (FPR; false rejections of clean inputs) under control.

Rather than detecting in the input space, we first pass the sample through the task backbone to obtain a latent embedding; a small binary MLP then learns to separate *clean* vs. *adversarial* points in this space. Let $e(x)$ be the embedding extracted from the task model and $g(\cdot)$ the detector; we compute a score $p = \sigma(g(e(x))) \in [0, 1]$ and *reject* if $p \geq \tau$. The threshold τ is calibrated to satisfy a target FPR (MLP scenario), or fixed ($\tau = 0.5$ for CNN). The overall workflow is illustrated in Figure below: the input (image or tabular) is mapped to an embedding by the task backbone; the detector predicts adversarial vs. clean; flagged inputs are rejected, otherwise they proceed to the task classifier.

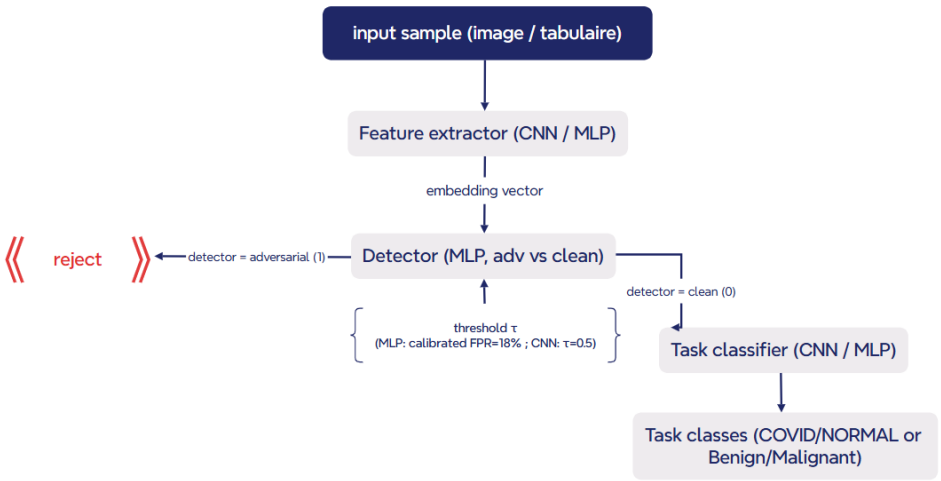


Figure 4.6: Detection of adversarial examples pipeline.

Usage protocol (shared)

- **Adversarial sets (ART):** generate train/test adversarial examples with FGSM, PGD, and BIM in all settings; add C&W (L2) at test time when specified by the scenario.
- **Embeddings:** pass inputs through the task model and extract a latent vector right before the final layer (scenario-specific details below).
- **Binary detector:** a small MLP consumes embeddings and outputs a single logit (*adversarial*= 1, *clean*= 0).

- **Training & selection:** train on the concatenation of clean and adversarial embeddings; track F1 each epoch; keep the best checkpoint.
- **Inference pipeline:** *embedding* → *detector* → reject if $p \geq \tau$, else forward to the task classifier.
- **Reported metrics:** (i) % of adversarial samples blocked; (ii) % of clean samples correctly classified (on accepted samples only); (iii) % of false rejections (clean blocked); (iv) breakdown by attack type.

Scenario A—Tabular (WDBC, MLP)

(a) Attacks & ART wrapper.

- **Train:** FGSM $\varepsilon = 0.2$; PGD $\varepsilon = 0.2$, steps = 10, $\varepsilon_{\text{step}} = 0.25\,\varepsilon$; BIM $\varepsilon = 0.2$, steps = 7, $\varepsilon_{\text{step}} = 0.10\,\varepsilon$.
- **Test:** FGSM $\varepsilon \in \{0.1, 0.2, 0.3\}$; PGD $\varepsilon \in \{0.1, 0.2\}$, steps = 20, $\varepsilon_{\text{step}} = 0.25\,\varepsilon$; BIM $\varepsilon \in \{0.1, 0.2\}$, steps = 10, $\varepsilon_{\text{step}} = 0.10\,\varepsilon$; C&W (L2) `initial_const` $\in \{0.1, 0.3\}$.
- **ART classifier:** `input_shape` $= (30,)$, `nb_classes` $= 2$, `clip_values` $= (-5.0, 5.0)$, no preprocessing (features are pre-standardized).

(b) Embeddings.

- **Extraction point:** after `fc3 + bn3 + ReLU`.
- **Normalization:** per-dimension z-score using (μ, σ) computed on *clean train* embeddings; apply the same (μ, σ) to all train/test (clean+adv) embeddings.

(c) Detector (architecture & training).

- **Architecture:** MLP [in → 256 → 128 → 1] with `BatchNorm1d` on hidden layers, ReLU, Dropout($p = 0.2$).
- **Loss:** `BCEWithLogitsLoss` with `pos_weight` = 2.0 (positive class = adversarial).
- **Optimizer:** AdamW (lr = 2×10^{-3}, weight decay = 10^{-4}).
- **Training length:** 35 epochs; **model selection:** best F1 on the detector eval set.

(d) Threshold calibration τ.

- Compute probabilities $p = \sigma(\text{logits})$ on the detector evaluation set; if AUC < 0.5, flip $p \leftarrow 1 - p$.
- Derive two thresholds: (i) F1-max and (ii) FPR-target from the ROC curve.
- **Pipeline threshold:** choose τ that achieves the highest TPR under the constraint FPR $\leq$ 0.18 (F1-max is reported but not used).

Scenario B—Imaging (COVID–Normal, CNN)

(a) Attacks & ART wrapper.

- **Train:** FGSM $\varepsilon = 4/255$; PGD $\varepsilon = 4/255$, steps = 10, $\varepsilon_{\text{step}} = 0.25\,\varepsilon$; BIM $\varepsilon = 4/255$, steps = 7, $\varepsilon_{\text{step}} = 0.10\,\varepsilon$ (no C&W at train time).
- **Test:** FGSM $\varepsilon \in \{4/255,\ 8/255\}$; PGD $\varepsilon \in \{4/255,\ 8/255\}$, steps = 40, $\varepsilon_{\text{step}} = 0.25\,\varepsilon$;

BIM $\varepsilon \in \{4/255,\ 8/255\}$, steps = 10, $\varepsilon_{\text{step}} = 0.10\,\varepsilon$; C&W (L2) `initial_const` = 0.3.

- **ART classifier:** `input_shape` = (3, IMG_SIZE, IMG_SIZE), `nb_classes` = 2, `clip_values` = (0, 1), `preprocessing` = (MEAN, STD) (ART applies $(x - \text{mean})/\text{std}$ before the model).

(b) Embeddings.

- **Extraction point:** output of `model.extract_features(x)` after input normalization $(x - \text{mean})/\text{std}$.
- **Normalization:** no additional z-score on embeddings in this scenario.

(c) Detector (architecture & training).

- **Architecture:** MLP [in $\to$ 128 $\to$ 1] with ReLU and Dropout($p = 0.2$) (no BatchNorm).
- **Loss:** `BCEWithLogitsLoss` (no `pos_weight`).
- **Optimizer:** AdamW (lr = 10^{-3}, weight decay = 10^{-4}).
- **Training length:** 20 epochs; **model selection:** best F1 on the detector eval set.

(d) Threshold τ.

- **Fixed:** $\tau = 0.5$ (no FPR-target calibration in this scenario).

4.5.5 Hybrid Defense: MAT + Defensive Distillation + Detector

Definition & principle. The hybrid defense proceeds in three steps. *First*, we make a strong teacher robust by *Mixed Adversarial Training (MAT)*: each mini-batch mixes clean and lightly perturbed samples so the model learns to remain stable under small, plausible changes. *Second*, we train a student to imitate the teacher via *defensive distillation*, using soft class probabilities at temperature T together with the ground-truth labels; at deployment we keep the student only. *Third*, we attach a small **detector** on top of the student's latent features to flag inputs that look adversarial and reject them before the final task decision.

We next instantiate this generic protocol in two settings: *Scenario—Tabular (WDBC, MLP)* and *Scenario—Imaging (COVID–19, CNN)*, where we detail the exact data spaces, attack grids, hyperparameters, and optimization choices for each.

Scenario 1—Imaging (COVID–19 Radiography, CNN)

Phase 1 — Mixed Adversarial Training (teacher).

- **Attack space & clamping:** perturb in the *normalized* space (after `Normalize`$(\boldsymbol{\mu}, \boldsymbol{\sigma})$); clamp to $\left[(0 - \boldsymbol{\mu})/\boldsymbol{\sigma},\ (1 - \boldsymbol{\mu})/\boldsymbol{\sigma}\right]$.
- **Budget/mix:** $\varepsilon_{\text{train}} = 4/255$, attacked fraction $\rho = 0.5$, loss blend weight $\lambda_{\text{adv}} = 0.3$.
- **Attack rotation per mini-batch:** FGSM (1) $\to$ PGD (3, $\alpha = \varepsilon/4$, random start) $\to$ BIM (5, $\alpha = \varepsilon/10$).
- **Optimization:** AdamW (lr = 10^{-3}, wd = 10^{-4}), up to 25 epochs, early stopping (patience = 5) on validation loss.

- **Output:** best checkpoint of the MAT-trained CNN (used as the *teacher*).

Phase 2 — Defensive Distillation (student).

- **Teacher:** MAT checkpoint from Phase 1, frozen (`eval()`, gradients disabled).
- **Student:** same SimpleCNN backbone (dropout 0.3).
- **Hybrid loss:** $\alpha T^2 \,\mathrm{KL}\big(p_t^{(T)} \,\|\, p_s^{(T)}\big) + (1-\alpha)\,\mathrm{CE}(z_s, y)$ with $T = 8.0$, $\alpha = 0.7$, $p^{(T)} = \mathrm{softmax}(z/T)$.
- **Optimization:** AdamW (lr = 10^{-3}, wd = 10^{-4}), early stopping (patience = 5).
- **Reference adversarial eval:** ART on images in $[0, 1]$ with preprocessing $(\boldsymbol{\mu}, \boldsymbol{\sigma})$; FGSM/PGD/BIM @ $\{2, 4, 8\}/255$ and C&W–L2 (e.g. $c_0 \in \{0.3, 0.5\}$).

Phase 3 — Detection & end-to-end pipeline (two versions).

Common steps (both versions).

- **Adversarial sets (ART).**
  - *Train (light grid):* FGSM/PGD/BIM @ $4/255$ (PGD: 10 steps, step = $\varepsilon/4$; BIM: 7 steps, step = $\varepsilon/10$).
  - *Test (full grid):* FGSM/PGD/BIM @ $\{4, 8\}/255$; C&W–L2 with `initial_const` = 0.3.
  - *ART config:* `clip_values` = $(0, 1)$, `preprocessing`= $(\boldsymbol{\mu}, \boldsymbol{\sigma})$.
- **Embeddings (from the student):** apply `Normalize`$(\boldsymbol{\mu}, \boldsymbol{\sigma})$, then extract `model.extract_features`; build detector datasets by concatenating clean/adv embeddings with labels $\{0, 1\}$.
- **Z-score:** compute (μ, σ) on *clean train* embeddings and apply to all (train/test; clean/adv).

Version 1 — Light detector, FPR target $\leq 10\%$.

- **Detector MLP:** $[\text{in} \to 128 \to 1]$, ReLU, Dropout($p = 0.2$).
- **Training:** BCEWithLogits, AdamW (lr = 10^{-3}, wd = 10^{-4}), 20 epochs; select best epoch by F1 (monitored at threshold 0.5).
- **Threshold calibration:** sweep probabilities on the detector test split; choose τ maximizing TPR under the constraint FPR ≤ 0.10.
- **Pipeline at inference:** embedding $\to$ detector (reject if $p \geq \tau$) $\to$ student classifier; report % adversarials blocked, % clean correctly classified (on accepted), % false rejections, and breakdown by attack.

Version 2 — Stronger detector (BN+Dropout), FPR target $\leq 15\%$.

- **Detector MLP:** $[\text{in} \to 256 \to 64 \to 1]$ with `BatchNorm1d` on hidden layers, ReLU, Dropout($p = 0.3$).
- **Training:** BCEWithLogits, AdamW (lr = 2×10^{-3}, wd = 10^{-4}), 20 epochs; select best epoch by F1.
- **Threshold calibration:** sweep probabilities; choose τ maximizing TPR with FPR ≤ 0.15.
- **Pipeline:** identical to Version 1; same set of KPIs reported.

Scenario 2—Tabular (WDBC, MLP)

Phase 1 — Mixed Adversarial Training (teacher).

- **Data space & clamping:** operate in the *standardized* (z-score) space; after each step clamp per feature using train-set bounds $[x_{\min}, x_{\max}]$ (computed on X_{train}).
- **Budget/mix:** $\varepsilon_{\text{train}} = 0.2$, attacked fraction $\rho = 0.5$, loss blend weight $\lambda_{\text{adv}} = 0.5$.
- **Attack rotation per mini-batch:** FGSM (1) → PGD (5, $\alpha = \varepsilon/5$, random start) → BIM (10, $\alpha = \varepsilon/10$).
- **Optimization:** AdamW (lr = 10^{-3}, wd = 10^{-4}), 30 epochs (no explicit early stopping).
- **Output:** best teacher checkpoint (MAT-trained MLP).

Phase 2 — Defensive Distillation (student).

- **Teacher:** Phase 1 MAT checkpoint, frozen (`eval()`, no gradients).
- **Student:** same MLP backbone.
- **Hybrid loss:** $\alpha T^2 \,\mathrm{KL}\big(p_t^{(T)} \,\|\, p_s^{(T)}\big) + (1-\alpha)\,\mathrm{CE}(z_s, y)$ with $T = 8.0$, $\alpha = 0.7$, $p^{(T)} = \mathrm{softmax}(z/T)$.
- **Optimization:** AdamW (lr = 10^{-3}, wd = 10^{-4}), 30 epochs.
- **Reference adversarial eval (student):** ART on standardized features with `clip_values` = $(-5, 5)$. Grids used for reporting:
  - FGSM @ $\varepsilon \in \{0.1, 0.2, 0.3\}$,
  - PGD (ℓ_∞) @ $\varepsilon \in \{0.1, 0.2, 0.3\}$, $\alpha = 0.01$, 20 iters,
  - BIM @ $\varepsilon \in \{0.1, 0.2, 0.3\}$, $\alpha = 0.01$, 10 iters,
  - C&W–L2: *FAST* (conf= 0, $c_0 = 0.3$, 75 iters, bsearch= 1, lr= 0.02) and *STRONG* (conf= 0, $c_0 = 0.01$, 500 iters, bsearch= 7, lr= 0.01).

Phase 3 — Detection & end-to-end pipeline (single version).

Adversarial sets (ART).

- *Train (light grid):* FGSM/PGD/BIM @ $\varepsilon = 0.2$ (PGD: 10 steps, step = $0.25\,\varepsilon$; BIM: 7 steps, step = $0.10\,\varepsilon$).
- *Test (full grid):* FGSM @ $\{0.1, 0.2, 0.3\}$; PGD @ $\{0.1, 0.2\}$ (20 steps, step = $0.25\,\varepsilon$); BIM @ $\{0.1, 0.2\}$ (10 steps, step = $0.10\,\varepsilon$); C&W–L2 with `initial_const` $\in \{0.1, 0.3\}$.
- *ART config:* `input_shape` = $(30,)$, `clip_values` = $(-5, 5)$, `preprocessing` = `None` (features already standardized).

Embeddings and normalization.

- **Extraction point (student):** hidden pathway `fc1/bn1` → `fc2/bn2` → `fc3/bn3` + ReLU (i.e., before the final classifier layer).
- **Z-score:** compute (μ, σ) on clean-train embeddings; apply the same (μ, σ) to all (train/test; clean/adv).

Binary detector (architecture & training).

- **MLP:** [in $\rightarrow$ 256 $\rightarrow$ 128 $\rightarrow$ 1] with `BatchNorm1d` on hidden layers, ReLU, Dropout(p = 0.2).
- **Loss/optimizer:** `BCEWithLogitsLoss` with class balancing (`pos_weight` from train counts), AdamW (lr = 2×10^{-3}, wd = 10^{-4}).
- **Schedule:** 35 epochs; select the best checkpoint by F1 on the detector test split (threshold 0.5 during monitoring).

Threshold calibration & pipeline metrics.

- **Calibration:** compute ROC on detector-test probabilities; choose τ that maximizes TPR under FPR ≤ 0.18 (after monotone flip if raw AUC < 0.5).
- **Inference pipeline:** embedding $\rightarrow$ detector (reject if $p \geq \tau$) $\rightarrow$ student classifier.
- **Reported KPIs:** % adversarial samples blocked; % clean samples correctly classified (on accepted only); % false rejections (clean blocked); breakdown by attack tag.

Chapter 5 Results and Analysis

Chapter Contents

5.1 Performance of the baseline (model without defense)

Before introducing defenses, we first quantify how vulnerable the unprotected models are. We evaluate two representative scenarios: a convolutional network on chest X-rays and a tabular MLP on WDBC. For each, we report the performance on clean data and under standard gradient-based adversarial perturbations. This establishes the reference level against which all subsequent defenses are compared.

5.1.1 Scenario A — Imaging (COVID–19 Radiography, CNN)

We consider a CNN trained in the usual supervised manner. On the clean test set it reaches strong accuracy, but its reliability degrades once inputs are exposed to small, human-imperceptible perturbations. We summarize the baseline performance on clean and adversarial samples with standard metrics (Accuracy, Precision, Recall, F1, ROC–AUC), and we complement the tables with score-vs-perturbation plots to make the trend visible. Overall, this scenario illustrates the typical sensitivity of image classifiers to bounded, structure-preserving distortions.

Illustration visuelle. Figure 5.1 makes the attack strength tangible: despite minimal perceptual change, the perturbed images can induce confident misclassification. Iterative methods (e.g., PGD/BIM) introduce faint, structured ripple patterns that are barely noticeable yet highly effective, while optimization-based C&W produces almost imperceptible changes that still flip the decision. As the perturbation budget increases, texture artifacts become slightly more visible and the baseline's errors grow sharply, highlighting how fragile the undefended CNN can be even under subtle input changes.

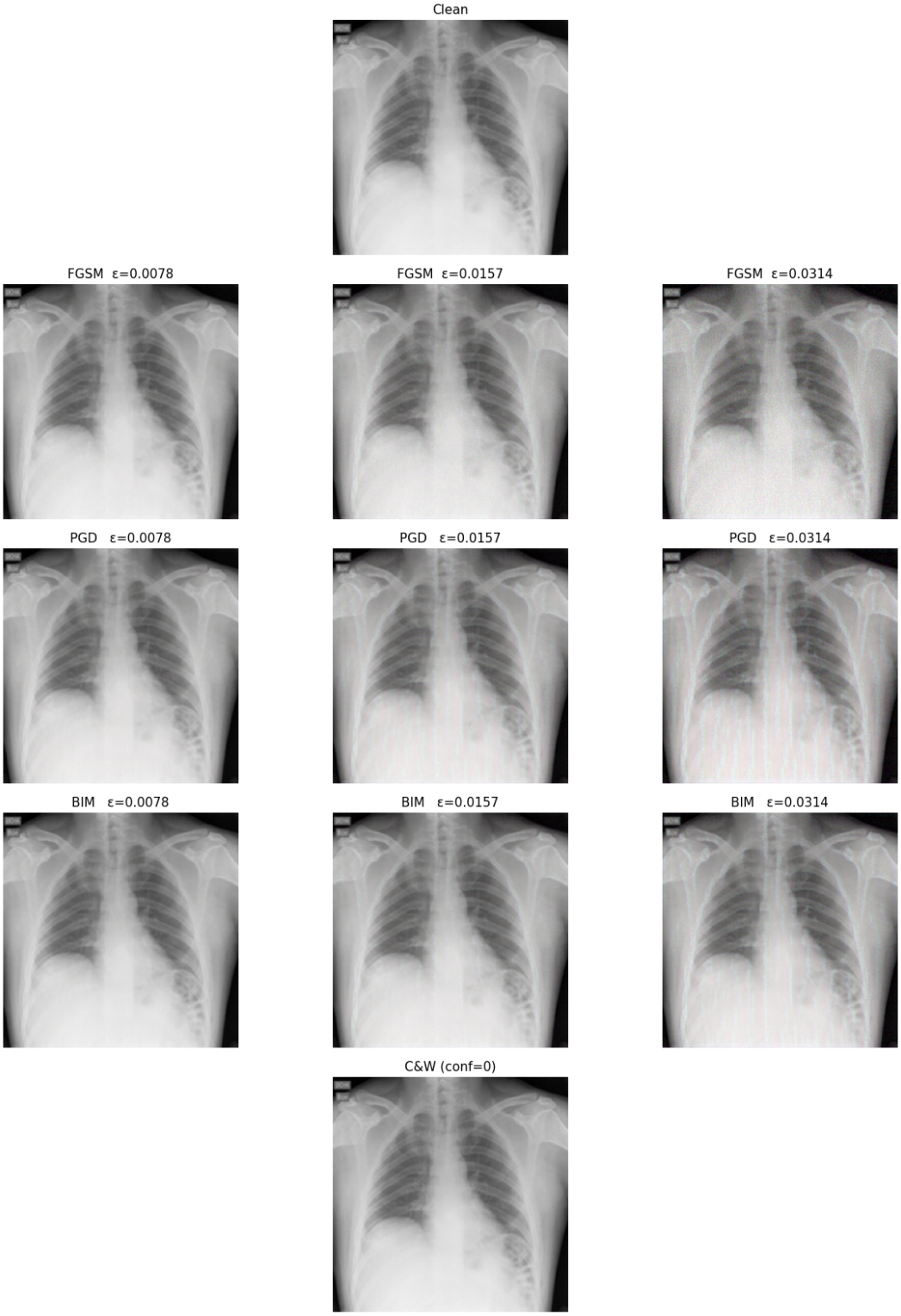


Figure 5.1: Clean image (top) and examples perturbed by different attacks.

Performance table and curves: clean model vs. attacks

Attack	ε / params	Acc.	Prec.	Rec.	F1	AUC
Clean	–	0.7900	0.7500	0.8700	0.8056	0.8380
FGSM	0.00784	0.4250	0.4490	0.6600	0.5344	0.4570
FGSM	0.01569	0.4150	0.4459	0.7000	0.5447	0.3130
FGSM	0.03137	0.4900	0.4949	0.9800	0.6577	0.2532
PGD	0.00784	0.2400	0.2400	0.2400	0.2400	0.2471
PGD	0.01569	0.2150	0.1647	0.1400	0.1514	0.2381
PGD	0.03137	0.2100	0.1548	0.1300	0.1413	0.2380
BIM	0.00784	0.2750	0.2936	0.3200	0.3062	0.2576
BIM	0.01569	0.2100	0.1705	0.1500	0.1596	0.2364
BIM	0.03137	0.2100	0.1548	0.1300	0.1413	0.2331
C&W–L2 (conf=0)	c_0=0.3	0.7600	0.7364	0.8100	0.7714	0.8313

Table 5.1: CNN baseline (undefended) on COVID-19 Radiography

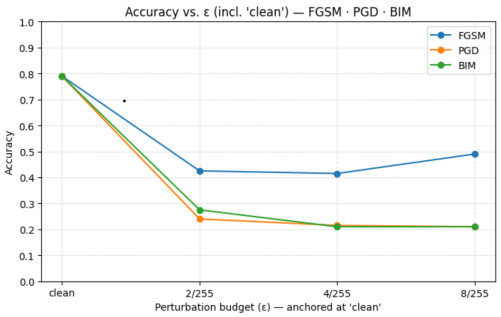


Figure 5.2: Accuracy vs. ε (incl. *clean*) — FGSM, PGD, BIM.

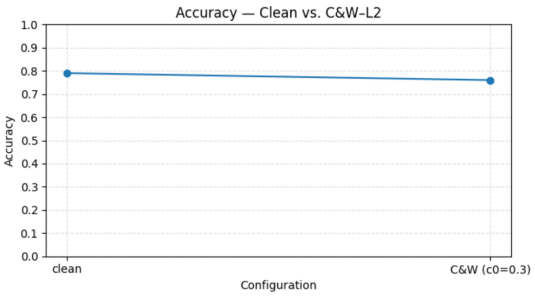


Figure 5.3: Accuracy — Clean vs. C&W–L2.

Interpretation. The table and curves jointly show that the undefended CNN is highly sensitive to ℓ_∞ attacks: PGD and BIM drive accuracy close to chance levels ($\approx$0.20–0.31) even at small budgets (2–8/255), while FGSM degrades performance substantially with a non-monotonic pattern typical of single-step attacks. In contrast, the chosen C&W–L2 configuration (c_0=0.3) leaves accuracy closer to the clean baseline (0.76 vs. 0.79), indicating that, for this budget, ℓ_2 perturbations are less damaging than ℓ_∞ ones—though still detrimental to reliability.

5.1.2 Scenario B — Tabular (WDBC, MLP)

We consider a standard MLP trained in the usual supervised manner on the WDBC tabular dataset (features are standardized). On the clean test split the model attains very strong performance, yet its reliability degrades as inputs are perturbed by small changes in the standardized space. As for the image scenario, we summarize baseline performance on clean and

adversarial samples with standard metrics (Accuracy, Precision, Recall, F1, ROC–AUC), and we complement the table with accuracy–vs–budget curves.

Performance table and curves: clean model vs. attacks

Attack	ε / params	Acc.	Prec.	Rec.	F1	AUC
Clean	–	0.9825	1.0000	0.9531	0.9760	0.9985
FGSM	0.10	0.9357	0.9818	0.8438	0.9076	0.9942
FGSM	0.20	0.8713	0.8750	0.7656	0.8167	0.9496
FGSM	0.30	0.7836	0.7368	0.6562	0.6942	0.8332
PGD	0.10, step 0.01, it 20	0.9357	0.9818	0.8438	0.9076	0.9939
PGD	0.20, step 0.02, it 20	0.8713	0.8750	0.7656	0.8167	0.9406
PGD	0.30, step 0.03, it 20	0.7544	0.6833	0.6406	0.6613	0.8014
BIM	0.10, step 0.01, it 10	0.9357	0.9818	0.8438	0.9076	0.9939
BIM	0.20, step 0.02, it 10	0.8713	0.8750	0.7656	0.8167	0.9467
BIM	0.30, step 0.03, it 10	0.7602	0.6949	0.6406	0.6667	0.8172
C&W–L2 (FAST)	conf= 0, c_0=0.3, it 75, bs 1	0.9240	0.9474	0.8438	0.8926	0.9947
C&W–L2 (STRONG)	conf= 0, c_0=0.01, it 500, bs 7	0.7953	0.7302	0.7188	0.7244	0.9501

Table 5.2: MLP baseline (undefended) on WDBC.

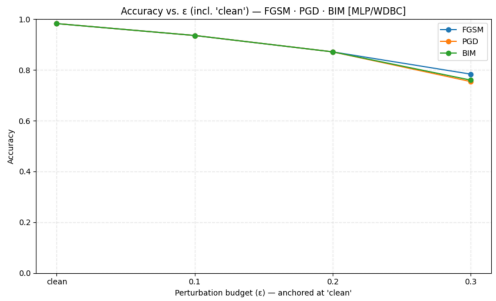


Figure 5.4: Accuracy vs. ε (incl. *clean*) — FGSM, PGD, BIM.

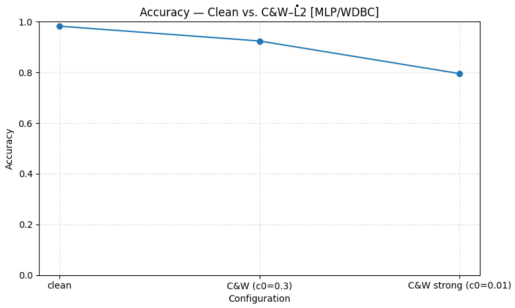


Figure 5.5: Accuracy — Clean vs. C&W–L2 (fast and strong).

Interpretation. The undefended MLP remains very accurate on clean data ($\approx$ 98.3%) but shows a monotonic degradation as ℓ_∞ budgets increase. Multi-step PGD/BIM slightly undercut FGSM at higher budgets (ε=0.3), yet the model still stays above 0.75 accuracy, reflecting the relative robustness often observed on low-dimensional, well-conditioned tabular features. For ℓ_2 attacks, C&W with a moderate c_0=0.3 keeps performance high ($\approx$ 92.4%), whereas a stronger configuration (c_0=0.01, it= 500) causes a larger drop ($\approx$ 79.5%), though still not as severe as strong ℓ_∞ settings.

5.2 Mixed Adversarial Training (MAT)

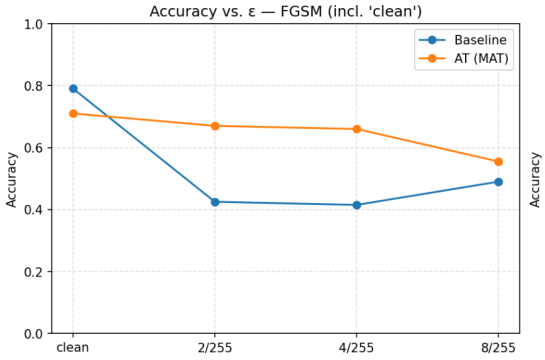
We train a *teacher* network with Mixed Adversarial Training (MAT): each mini-batch mixes clean samples with on-the-fly adversarial counterparts (FGSM/PGD/BIM) crafted under a bounded budget ε in the same space used for learning (normalized images for CNNs, standardized features for MLPs). The loss blends clean and adversarial cross-entropy with weight λ_{adv}, encouraging features that remain stable under small, worst-case perturbations. This protocol is identical across modalities and thus serves as a common starting point for the two case studies below: **Scenario A** (imaging, CNN) and **Scenario B** (tabular, MLP). The MAT checkpoint will be used as the *teacher* for defensive distillation and as the backbone from which we extract embeddings for the detector.

5.2.1 Scenario A — Imaging (COVID–19 Radiography, CNN)

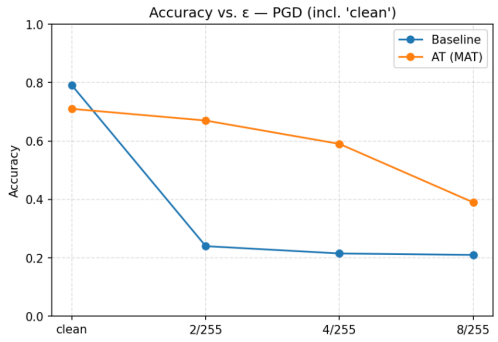
Performance table and curves: defended model (MAT) vs. attacks

Attack	ε / params	Acc.	Prec.	Rec.	F1	AUC
Clean (MAT)	–	0.7100	0.6780	0.8000	0.7339	0.7622
FGSM	0.00784	0.6700	0.6466	0.7500	0.6944	0.7483
FGSM	0.01569	0.6600	0.6633	0.6500	0.6566	0.7001
FGSM	0.03137	0.5550	0.5679	0.4600	0.5083	0.5625
PGD	0.00784	0.6700	0.6518	0.7300	0.6887	0.7387
PGD	0.01569	0.5900	0.5938	0.5700	0.5816	0.6310
PGD	0.03137	0.3900	0.3514	0.2600	0.2989	0.3734
BIM	0.00784	0.6700	0.6466	0.7500	0.6944	0.7438
BIM	0.01569	0.6200	0.6224	0.6100	0.6162	0.6600
BIM	0.03137	0.4300	0.4146	0.3400	0.3736	0.4240
C&W–L2 (conf=0)	c_0=0.3	0.7050	0.6752	0.7900	0.7281	0.7622

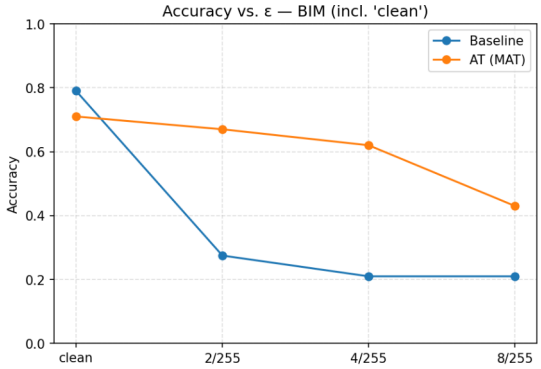
Table 5.3: CNN with Mixed Adversarial Training (MAT): clean vs. adversarial inputs



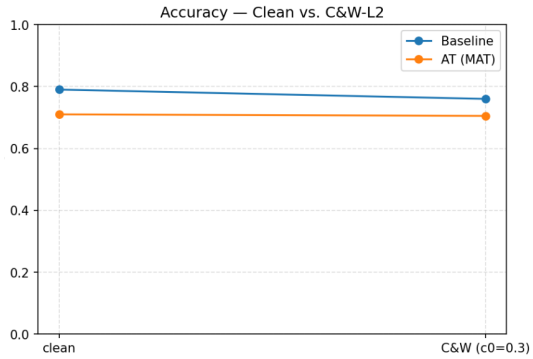
(a) FGSM — Accuracy vs. ε (ancré à *clean*).



(b) PGD — Accuracy vs. ε (ancré à *clean*).



(c) BIM — Accuracy vs. ε (ancré à *clean*).



(d) C&W–L2 — Accuracy: clean vs. C&W.

Figure 5.6: Baseline (non défendu) vs. MAT (défendu) — courbes d'accuracy par attaque.

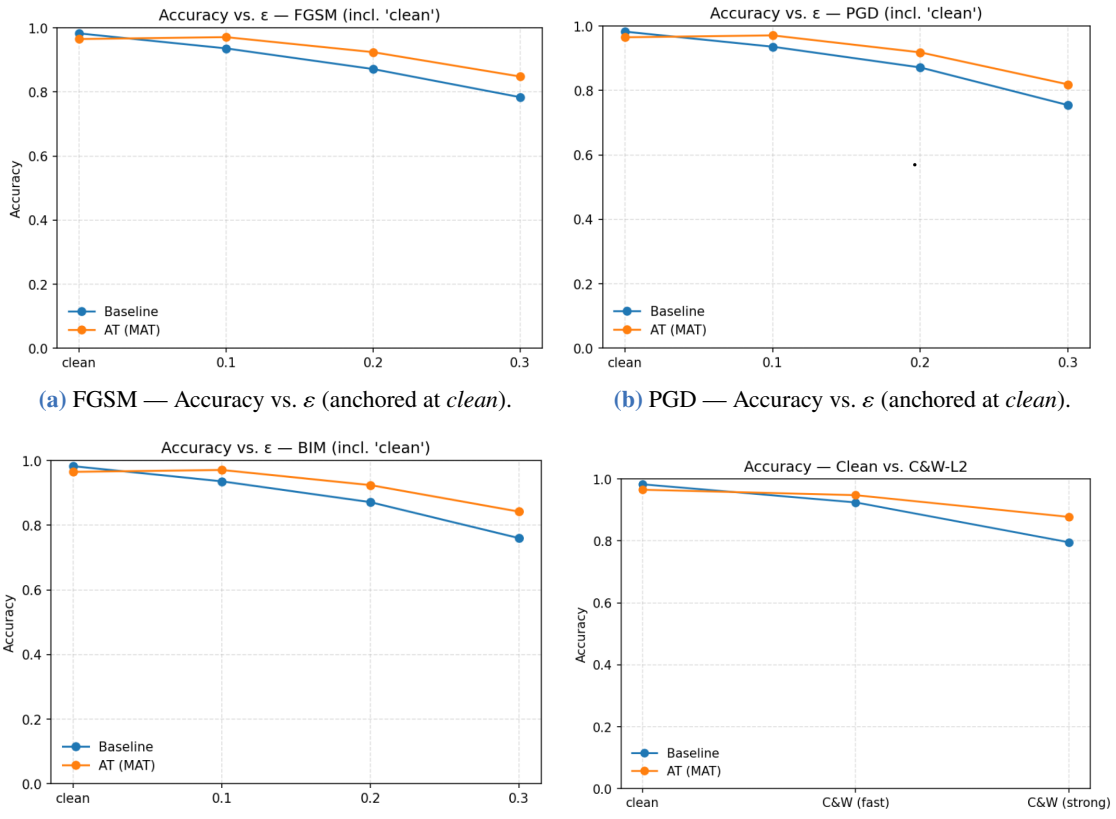
Interpretation. Compared to the undefended baseline, MAT markedly improves robustness to ℓ_∞ attacks at moderate budgets: e.g., at $2/255$, accuracy rises from $0.42/0.24/0.28$ (FGSM/PGD/BIM) to ≈ 0.67 across methods. Gains persist at $4/255$, while performance still drops at $8/255$, reflecting the usual trade-off between robustness and accuracy (clean accuracy decreases from 0.79 to 0.71). For ℓ_2 C&W with c_0=0.3, the defended model stays close to clean performance (0.705 vs. 0.710), indicating that, under this configuration, MAT primarily strengthens resistance to ℓ_∞-bounded perturbations.

5.2.2 Scenario B — Tabular (WDBC, MLP)

Performance table and curves: defended model (MAT) vs. attacks

Attack	ε / params	Acc.	Prec.	Rec.	F1	AUC
Clean (MAT)	–	0.9649	1.0000	0.9062	0.9508	0.9987
FGSM	0.10	0.9708	0.9836	0.9375	0.9600	0.9965
FGSM	0.20	0.9240	0.9322	0.8594	0.8943	0.9775
FGSM	0.30	0.8480	0.8065	0.7812	0.7937	0.9190
PGD	0.10, step 0.01, it 20	0.9708	0.9836	0.9375	0.9600	0.9963
PGD	0.20, step 0.02, it 20	0.9181	0.9167	0.8594	0.8871	0.9734
PGD	0.30, step 0.03, it 20	0.8187	0.7619	0.7500	0.7559	0.9007
BIM	0.10, step 0.01, it 10	0.9708	0.9836	0.9375	0.9600	0.9965
BIM	0.20, step 0.02, it 10	0.9240	0.9322	0.8594	0.8943	0.9755
BIM	0.30, step 0.03, it 10	0.8421	0.8033	0.7656	0.7840	0.9092
C&W–L2 (FAST)	conf= 0, c_0=0.3, it 75, bs 1	0.9474	0.9825	0.8750	0.9256	0.9974
C&W–L2 (STRONG)	conf= 0, c_0=0.01, it 500, bs 7	0.8772	0.8308	0.8438	0.8372	0.9766

Table 5.4: MLP with Mixed Adversarial Training (MAT): clean vs. adversarial inputs



(a) FGSM — Accuracy vs. ε (anchored at *clean*).

(b) PGD — Accuracy vs. ε (anchored at *clean*).

(c) BIM — Accuracy vs. ε (anchored at *clean*).

(d) C&W–L2 — Accuracy: clean vs. C&W (fast/strong).

Figure 5.7: Baseline (undefended) vs. MAT (defended) — accuracy curves per attack.

Interpretation. MAT preserves high clean accuracy ($\approx$96.5%) and yields consistent robustness gains across ℓ_∞ attacks: at ε=0.30, the defended MLP remains around 0.85 (FGSM), 0.82 (PGD), and 0.84 (BIM) accuracy, markedly above the corresponding undefended baselines.

For ℓ_2 perturbations, the fast C&W configuration keeps performance near 0.95, while a strong configuration still maintains ≈ 0.88, illustrating a favorable robustness/accuracy trade-off under MAT.

5.3 Defensive Distillation:

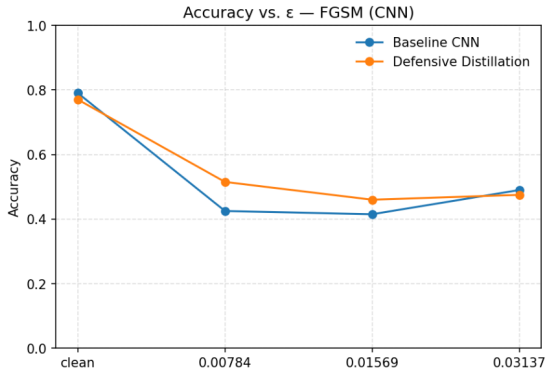
We train a *student* via *Defensive Distillation*, using a frozen teacher to provide softened targets; the student is then evaluated on clean and adversarial inputs. As in the MAT section, we report standard metrics (Accuracy, Precision, Recall, F1, ROC–AUC) and include accuracy–vs–budget curves per attack. This section focuses on the distilled student alone .

5.3.1 Scenario A — Imaging (COVID–19 Radiography, CNN)

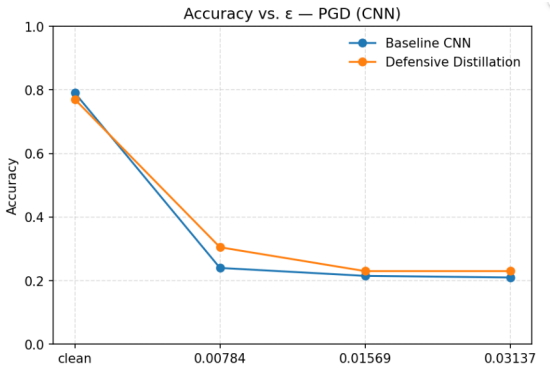
Performance table and curves: distilled model vs. attacks

Attack	ε / params	Acc.	Prec.	Rec.	F1	AUC
Clean (Distill)	–	0.7700	0.7077	0.9200	0.8000	0.8094
FGSM	0.00784	0.5150	0.5102	0.7500	0.6073	0.5321
FGSM	0.01569	0.4600	0.4718	0.6700	0.5537	0.3769
FGSM	0.03137	0.4750	0.4865	0.9000	0.6316	0.2871
PGD	0.00784	0.3050	0.2990	0.2900	0.2944	0.2831
PGD	0.01569	0.2300	0.1143	0.0800	0.0941	0.2742
PGD	0.03137	0.2300	0.1143	0.0800	0.0941	0.2781
BIM	0.00784	0.3400	0.3519	0.3800	0.3654	0.3112
BIM	0.01569	0.2250	0.1333	0.1000	0.1143	0.2717
BIM	0.03137	0.2300	0.1143	0.0800	0.0941	0.2773
C&W–L2 (conf=0)	c_0=0.3	0.7350	0.6880	0.8600	0.7644	0.8065

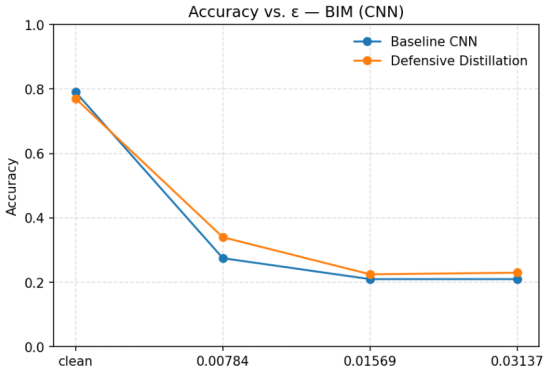
Table 5.5: CNN with Defensive Distillation: clean vs. adversarial inputs.



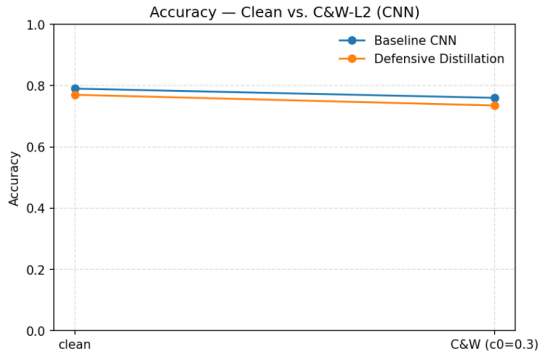
(a) FGSM — Accuracy vs. ε (anchored at *clean*).



(b) PGD — Accuracy vs. ε (anchored at *clean*).



(c) BIM — Accuracy vs. ε (anchored at *clean*).



(d) C&W–L2 — Accuracy: clean vs. C&W.

Figure 5.8: Baseline (undefended) vs. Defensive Distillation (defended) — accuracy curves per attack.

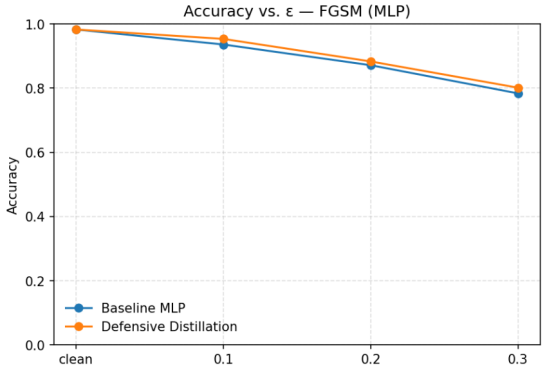
Interpretation. Distillation slightly lowers clean accuracy compared to the baseline (0.77 vs. 0.79) but yields modest gains against ℓ_∞ attacks at small budgets: e.g., at $2/255$, accuracy improves from $0.43/0.24/0.28$ (FGSM/PGD/BIM) to $\approx 0.52/0.31/0.34$. Improvements taper at larger budgets and remain limited for multi-step PGD/BIM, while performance under C&W–L2 (c_0=0.3) is close to but slightly below clean (0.735 vs. 0.770). Overall, distillation provides partial robustness benefits without the stronger gains observed with MAT.

5.3.2 Scenario B — Tabular (WDBC, MLP)

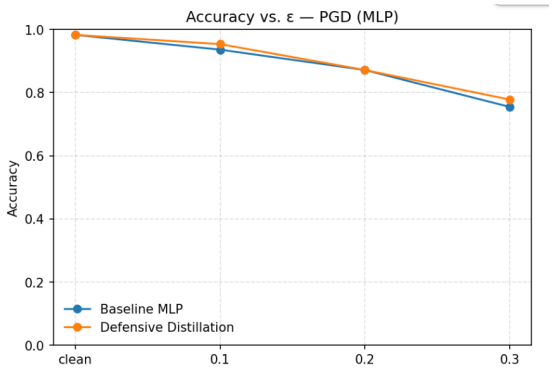
Performance table and curves: distilled model vs. attacks

Attack	ε / params	Acc.	Prec.	Rec.	F1	AUC
Clean (Distill)	–	0.9825	0.9841	0.9688	0.9764	0.9978
FGSM	0.10	0.9532	0.9828	0.8906	0.9344	0.9950
FGSM	0.20	0.8830	0.8438	0.8438	0.8438	0.9619
FGSM	0.30	0.8012	0.7344	0.7344	0.7344	0.8769
PGD	0.10, step 0.01, it 20	0.9532	0.9828	0.8906	0.9344	0.9947
PGD	0.20, step 0.02, it 20	0.8713	0.8182	0.8438	0.8308	0.9546
PGD	0.30, step 0.03, it 20	0.7778	0.6912	0.7344	0.7121	0.8490
BIM	0.10, step 0.01, it 10	0.9532	0.9828	0.8906	0.9344	0.9949
BIM	0.20, step 0.02, it 10	0.8772	0.8308	0.8438	0.8372	0.9588
BIM	0.30, step 0.03, it 10	0.7836	0.7015	0.7344	0.7176	0.8640
C&W–L2 (FAST)	conf= 0, c_0=0.3, it 75, bs 1, lr 0.02	0.9240	0.9180	0.8750	0.8960	0.9939
C&W–L2 (STRONG)	conf= 0, c_0=0.01, it 500, bs 7, lr 0.01	0.8363	0.7500	0.8438	0.7941	0.9683

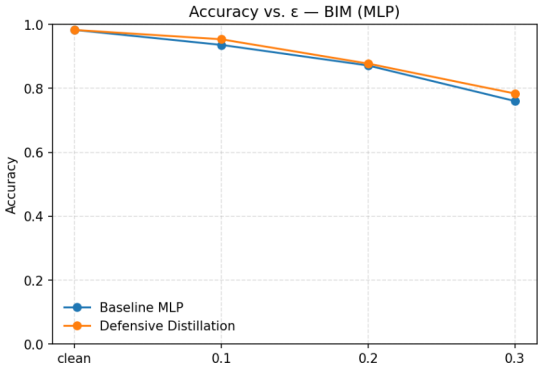
Table 5.6: MLP with Defensive Distillation on WDBC: clean vs. adversarial inputs .



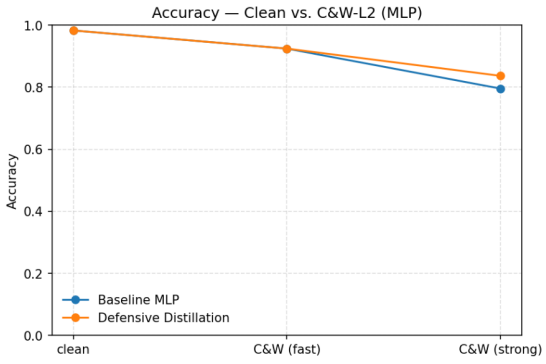
(a) FGSM — Accuracy vs. ε (anchored at *clean*).



(b) PGD — Accuracy vs. ε (anchored at *clean*).



(c) BIM — Accuracy vs. ε (anchored at *clean*).



(d) C&W–L2 — Accuracy: clean vs. C&W (fast/strong).

Figure 5.9: Baseline (undefended) vs. Defensive Distillation (defended, MLP/WDBC) — accuracy curves per attack.

Interpretation. On WDBC, defensive distillation preserves clean accuracy ($\approx$ 98.3%) and yields small but consistent robustness gains over the baseline across ℓ_∞ budgets: at ε=0.1/0.2/0.3,

accuracy improves for FGSM (0.953/0.883/0.801), PGD (0.953/ 0.871/0.778), and BIM (0.953/0.877/0.784). For ℓ_2 attacks, the distilled model matches the baseline under the *fast* C&W (0.924) and clearly outperforms it under the *strong* setting (0.836 vs. 0.795). Overall, on tabular features, distillation provides measurable robustness gains with negligible clean-performance cost.

5.4 Random Noise Injection:

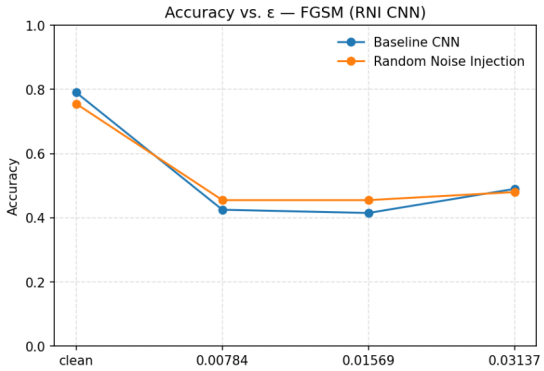
We evaluate a simple *Random Noise Injection* (RNI) defense in which small, isotropic noise is added to dampen gradient-aligned perturbations. As in prior scenarios, we report standard metrics (Accuracy, Precision, Recall, F1, ROC–AUC) for clean and adversarial inputs, and we complement the table with accuracy–vs–budget curves per attack.

5.4.1 Scenario A — Imaging (COVID–19 Radiography, CNN)

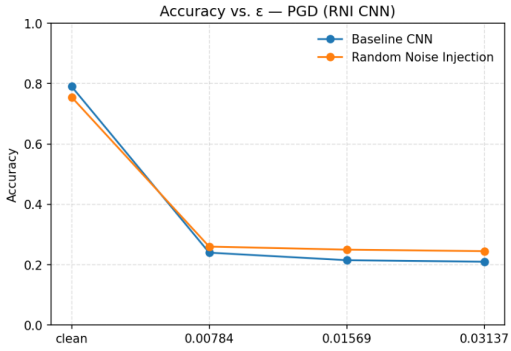
Performance table and curves: defended model (RNI) vs. attacks

Attack	ε / params	Acc.	Prec.	Rec.	F1	AUC
Clean (RNI)	–	0.7550	0.7931	0.6900	0.7380	0.8519
FGSM	0.00784	0.4550	0.4702	0.7100	0.5657	0.4305
FGSM	0.01569	0.4550	0.4743	0.8300	0.6036	0.3278
FGSM	0.03137	0.4800	0.4898	0.9600	0.6486	0.2692
PGD	0.00784	0.2600	0.2895	0.3300	0.3084	0.3360
PGD	0.01569	0.2500	0.2807	0.3200	0.2991	0.2954
PGD	0.03137	0.2450	0.2743	0.3100	0.2911	0.2566
BIM	0.00784	0.2700	0.3017	0.3500	0.3241	0.3404
BIM	0.01569	0.2500	0.2807	0.3200	0.2991	0.3006
BIM	0.03137	0.2450	0.2743	0.3100	0.2911	0.2542
C&W–L2 (conf=0)	c_0=0.3	0.7250	0.7647	0.6500	0.7027	0.8490

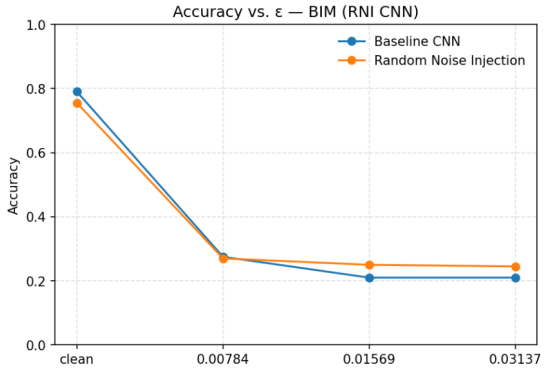
Table 5.7: CNN with Random Noise Injection (RNI): clean vs. adversarial inputs.



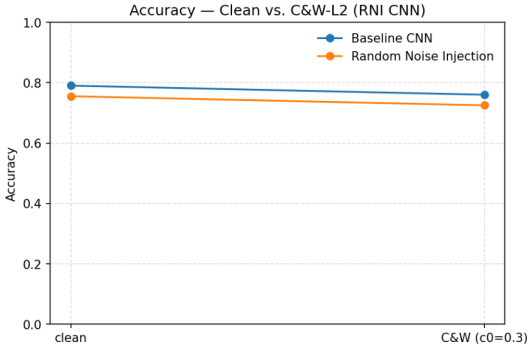
(a) FGSM — Accuracy vs. ε (anchored at *clean*).



(b) PGD — Accuracy vs. ε (anchored at *clean*).



(c) BIM — Accuracy vs. ε (anchored at *clean*).



(d) C&W–L2 — Accuracy: clean vs. C&W.

Figure 5.10: Baseline (undefended) vs. RNI (defended) — accuracy curves per attack.

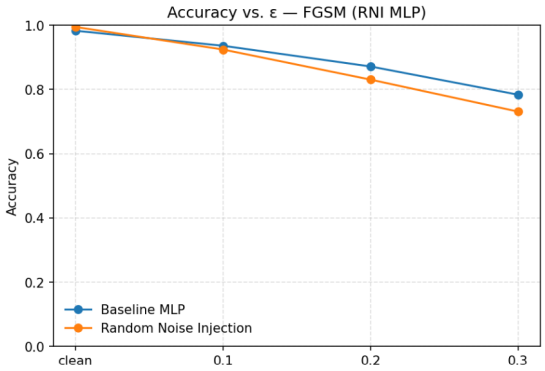
Interpretation. RNI incurs a modest clean-accuracy drop (0.79 → 0.755) but slightly improves resilience against several attack settings. Gains are most visible against iterative ℓ_∞ attacks at small budgets (e.g., PGD/BIM at $2/255$ show higher AUC than baseline), and FGSM improves at $2/255$–$4/255$ in F1/Acc. However, performance remains low under stronger ℓ_∞ budgets (e.g., $8/255$) and only approaches clean levels for the tested ℓ_2 C&W configuration. Overall, RNI provides limited hardening compared to MAT, trading a bit of clean accuracy for small robustness gains that are insufficient against strong iterative attacks.

5.4.2 Scenario B — Tabular (WDBC, MLP)

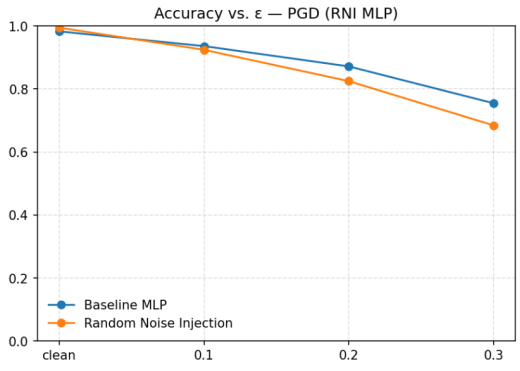
Performance table and curves: defended model (RNI) vs. attacks

Attack	ε / params	Acc.	Prec.	Rec.	F1	AUC
Clean (RNI)	–	0.9942	1.0000	0.9844	0.9921	0.9990
FGSM	0.10	0.9240	0.9180	0.8750	0.8960	0.9895
FGSM	0.20	0.8304	0.7778	0.7656	0.7717	0.9216
FGSM	0.30	0.7310	0.6324	0.6719	0.6515	0.7824
PGD	0.10, step 0.01, it 20	0.9240	0.9180	0.8750	0.8960	0.9893
PGD	0.20, step 0.02, it 20	0.8246	0.7656	0.7656	0.7656	0.9083
PGD	0.30, step 0.03, it 20	0.6842	0.5694	0.6406	0.6029	0.7503
BIM	0.10, step 0.01, it 10	0.9240	0.9180	0.8750	0.8960	0.9893
BIM	0.20, step 0.02, it 10	0.8246	0.7656	0.7656	0.7656	0.9144
BIM	0.30, step 0.03, it 10	0.6959	0.5833	0.6562	0.6176	0.7647
C&W–L2 (FAST)	conf= 0, c_0=0.3, it 75, bs 1, lr 0.02	0.9181	0.8906	0.8906	0.8906	0.9926

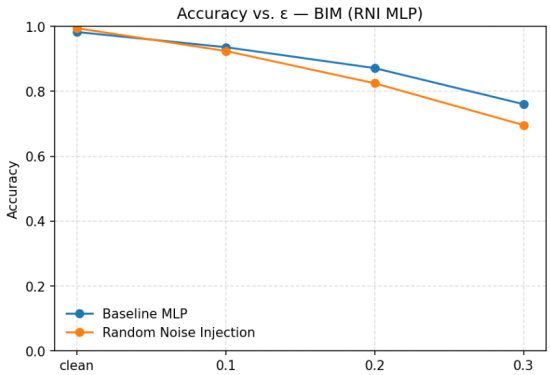
Table 5.8: MLP with Random Noise Injection (RNI): clean vs. adversarial inputs .



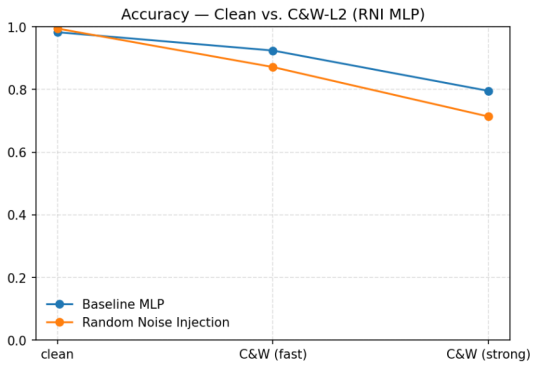
(a) FGSM — Accuracy vs. ε (anchored at *clean*).



(b) PGD — Accuracy vs. ε (anchored at *clean*).



(c) BIM — Accuracy vs. ε (anchored at *clean*).



(d) C&W–L2 — Accuracy: clean vs. C&W.

Figure 5.11: Baseline (undefended) vs. RNI (defended) — accuracy curves per attack.

Interpretation. On WDBC, RNI *raises* clean accuracy slightly (from ≈ 0.983 to 0.994), but it does *not* translate into stronger ℓ_∞ robustness: across FGSM/PGD/BIM, accuracy is consistently lower than the baseline as ε grows (e.g., at ε=0.3, 0.73/0.68/0.70 vs. 0.78/0.75/0.76). The C&W–L2 (fast) setting remains relatively high (≈ 0.918), consistent with the idea that small

isotropic noise helps little against iterative ℓ_∞ attacks on low-dimensional tabular features. Overall, RNI acts more as a clean-data regularizer here than a robustness booster, and is less effective than other methodes for this scenario.

5.5 Detection OF Adversarial Examples:

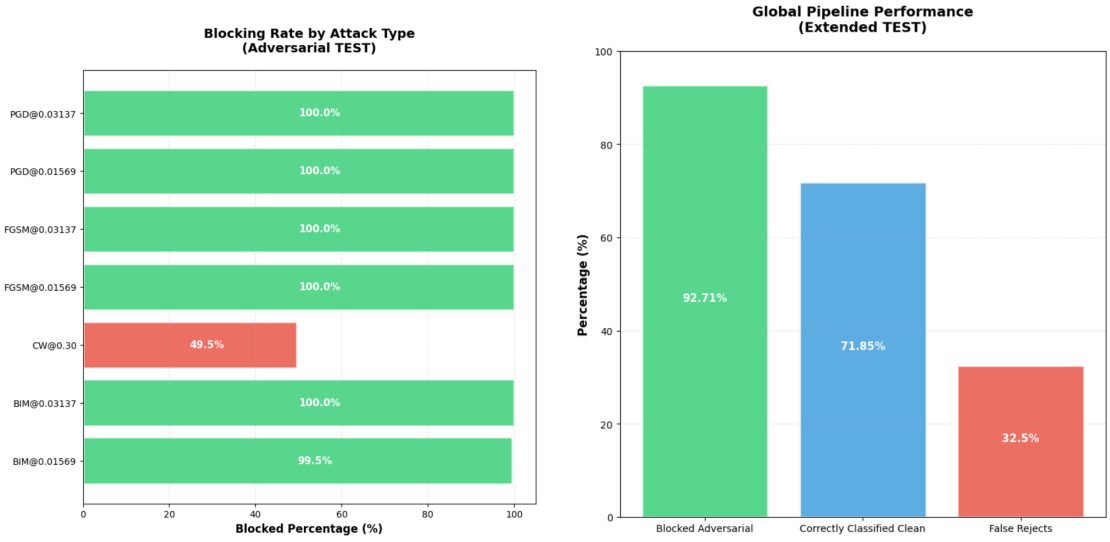
Here we evaluate an adversarial-example detector placed before the classifier. We present its standalone metrics and system-level impact (adversarial blocking rate, clean acceptance, false rejects), first on imaging (Scenario A — CNN) and then on tabular data (Scenario B — MLP).

5.5.1 Scenario A — Imaging (COVID–19 Radiography, CNN)

Accuracy	Precision	Recall (TPR)	F1	AUC
0.896	0.952	0.927	0.940	0.941

Table 5.9: Detector-only metrics on the clean/adversarial test split .

Detector-only performance (test split).



(a) Blocking rate by attack type on the adversarial test set. **(b)** Global pipeline KPIs on the extended test.

Figure 5.12: Detector in the end-to-end pipeline: per-attack blocking (left) and overall KPIs (right).

Interpretation. The detector achieves strong separation on the test split (AUC $\approx$ 0.94) and blocks most ℓ_∞ attacks (FGSM/PGD/BIM) at the evaluated budgets (close to 100%), but is less effective against the optimization-based C&W–L2 configuration ($\sim$49.5% blocked). In the full pipeline, 92.7% of adversarial inputs are rejected, yet the false-reject rate on clean images remains high (32.5%), yielding only 71.9% correct classification among accepted clean samples. This highlights the sensitivity of threshold τ and the need to calibrate it under an explicit FPR

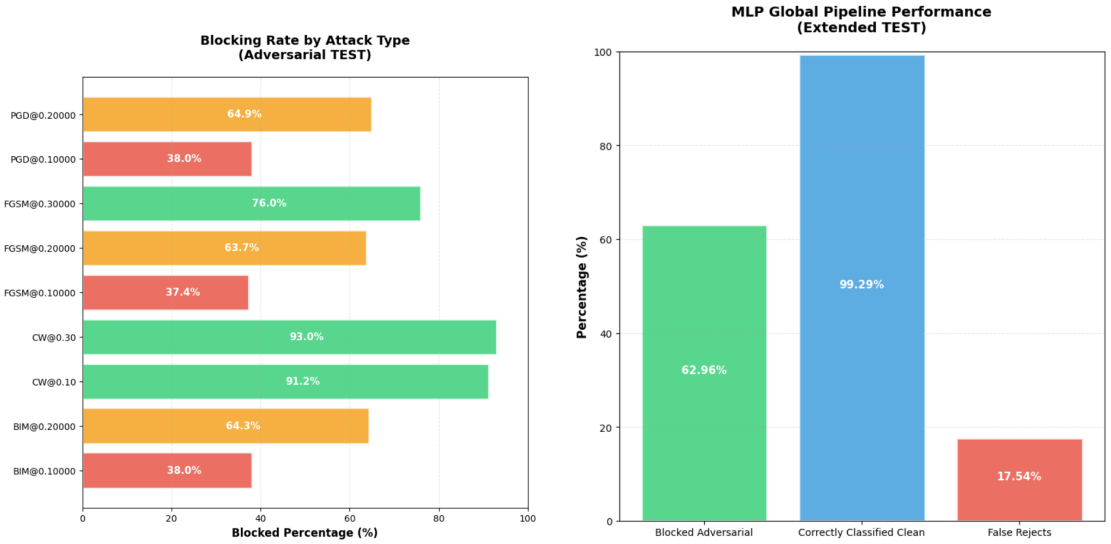
target and to enrich the detector training with ℓ_2 attacks to improve coverage without unduly rejecting clean inputs.

5.5.2 Scenario B — Tabular (WDBC, MLP):

Detector performance (test).

Model	Acc.	Prec.	Rec.	F1	AUC
Detector (MLP)	0.649	0.970	0.630	0.764	0.824

Table 5.10: Binary detector on MLP embeddings (adversarial vs. clean).



(a) Blocking rate by attack type (adversarial test).

(b) Global pipeline: blocked adv., accepted clean, false rejects.

Figure 5.13: MLP detector: breakdown of blocked adversarials and end-to-end KPIs.

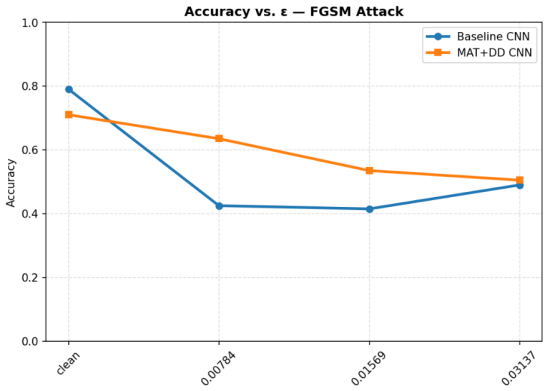
Interpretation. The detector attains high precision ($\approx$0.97) but moderate recall ($\approx$0.63), which limits the overall blocked-adversarial rate (panel 5.13a and 5.13b). Blocking is strong on ℓ_2 C&W (about 91–93%) and increases with the perturbation budget for ℓ_∞ attacks (FGSM/PGD/BIM: $\sim$ 37% at ε=0.1 up to $\sim$ 64–76% at higher ε). End-to-end, the pipeline blocks $\sim$ 63% of adversarial inputs while preserving $\sim$99.3% accuracy on accepted clean samples, at the cost of $\sim$ 17.5% false rejects. Adjusting the decision threshold τ along the ROC curve can trade off higher blocking for lower clean rejection depending on deployment priorities.

5.6 Hybrid Defense: MAT + Defensive Distillation + Detector:

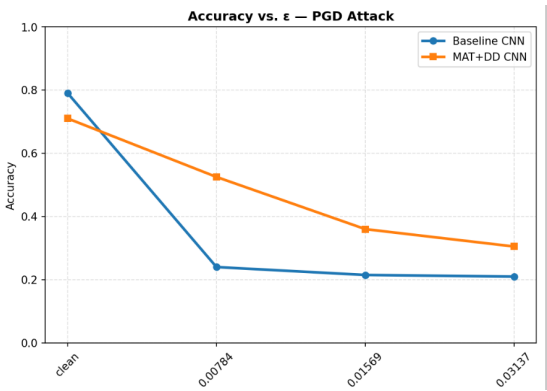
We deploy a *hybrid* pipeline: a CNN teacher trained with Mixed Adversarial Training (MAT), a distilled student (soft targets, temperature scaling), and a lightweight detector operating on student embeddings. We report the student's robustness curves and the detector's standalone and end-to-end performance.

5.6.1 Scenario A — Imaging (COVID–19 Radiography, CNN)

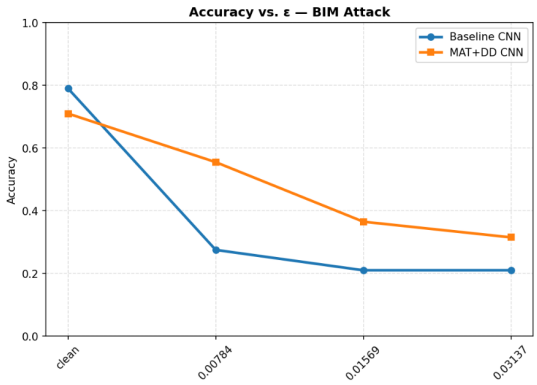
5.6.1.1 Version1:



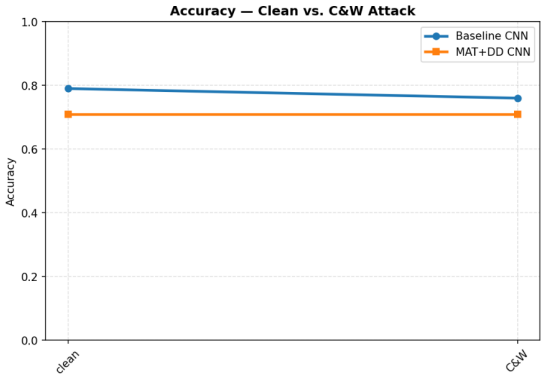
(a) FGSM — Accuracy vs. ε (anchored at *clean*).



(b) PGD — Accuracy vs. ε (anchored at *clean*).

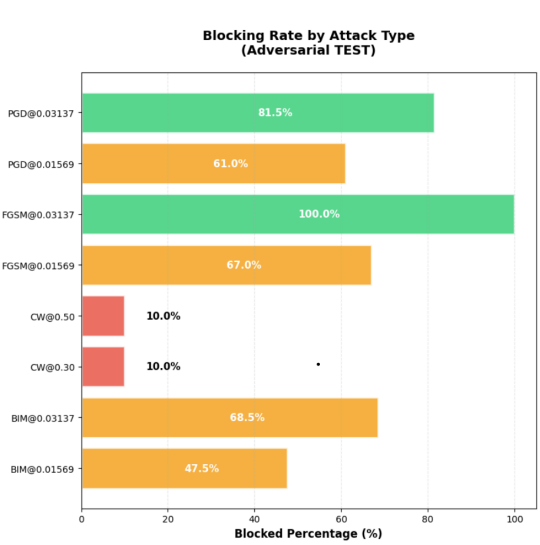


(c) BIM — Accuracy vs. ε (anchored at *clean*).

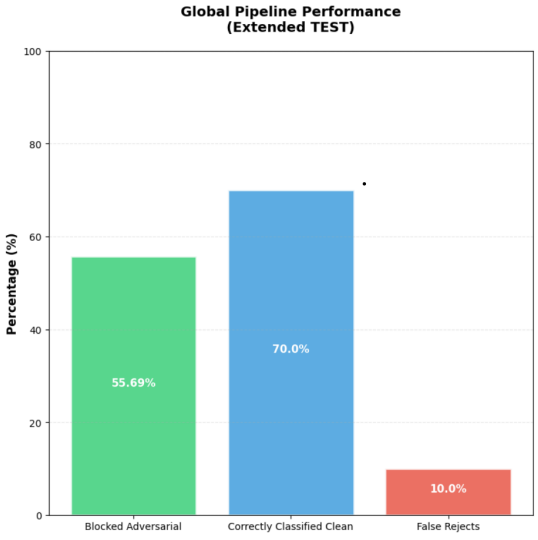


(d) C&W–L2 — Accuracy: clean vs. C&W.

Figure 5.14: Student (MAT → Distillation) — robustness curves per attack.



(a) Blocking rate by attack type (adversarial test).

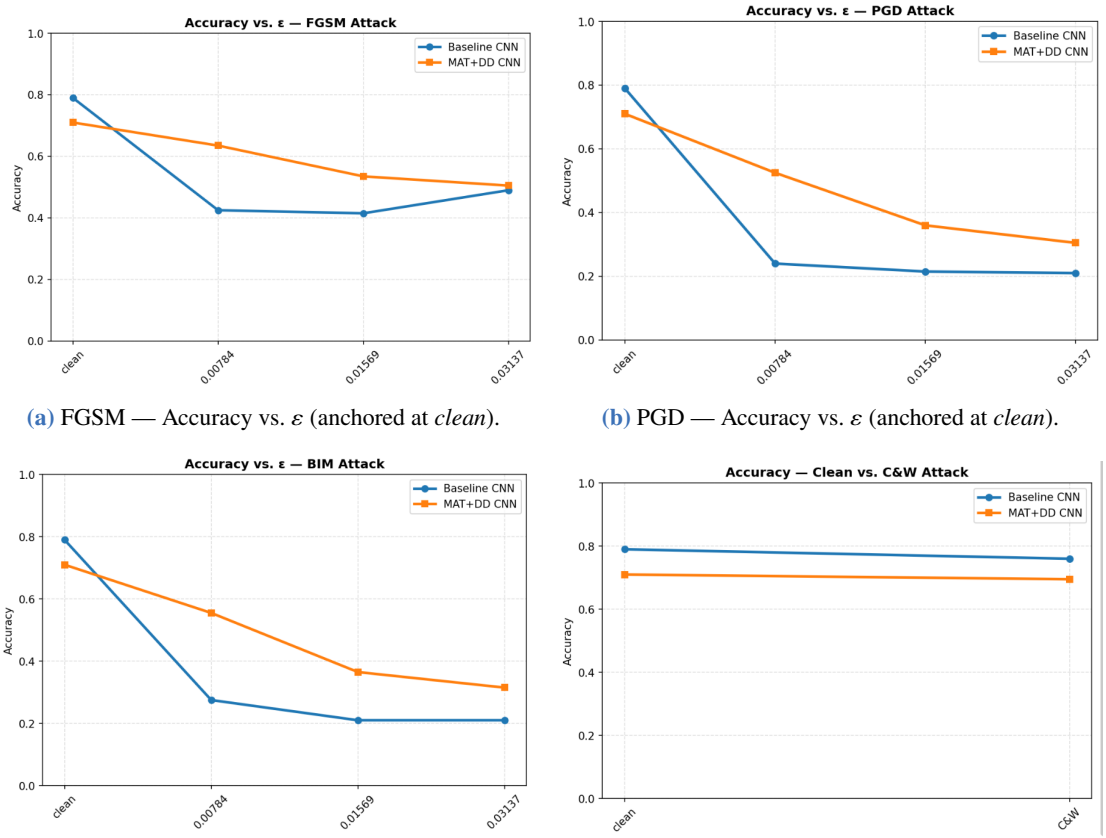


(b) Global pipeline (blocked adv., clean kept, false rejects).

Figure 5.15: Detector effectiveness and end-to-end impact.

Interpretation. The MAT → distilled student improves over the undefended baseline at small to moderate ℓ_∞ budgets (e.g., FGSM/PGD/BIM at $2/255$–$4/255$) but still degrades for stronger iterative settings ($8/255$). The detector blocks moderatly overall ($\sim 55\%$), with moderate rates on PGD/BIM/FGSM and lower sensitivity to C&W (about 15%). End-to-end, the pipeline preserves a majority of clean decisions ($\sim 70\%$) at the cost of non-negligible false rejects ($\sim 10\%$), reflecting the classic robustness–utility trade-off.

5.6.1.2 Version2:



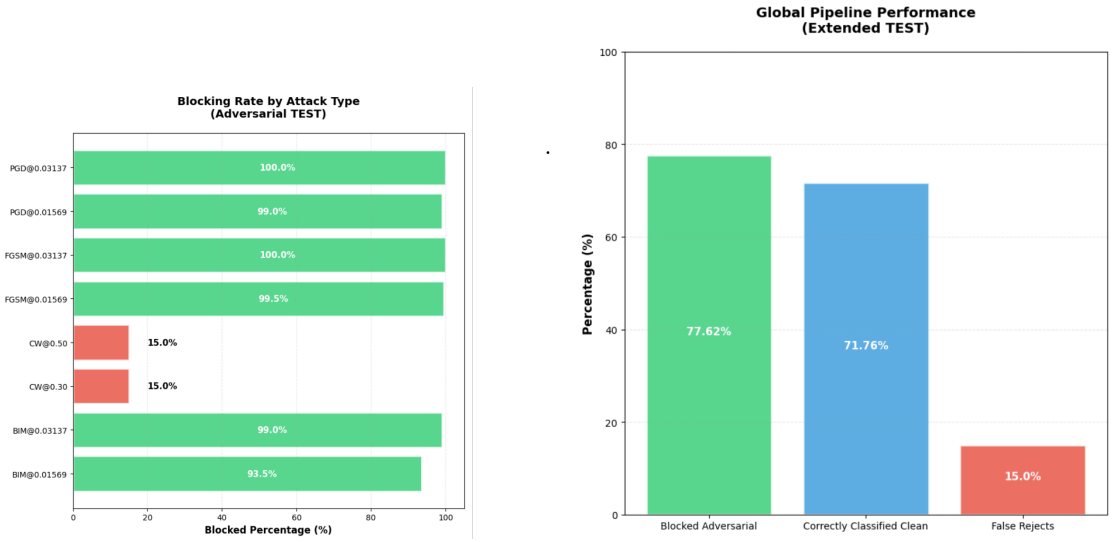
(a) FGSM — Accuracy vs. ε (anchored at *clean*).

(b) PGD — Accuracy vs. ε (anchored at *clean*).

(c) BIM — Accuracy vs. ε (anchored at *clean*).

(d) C&W–L2 — Accuracy: clean vs. C&W.

Figure 5.16: Student (MAT → Distillation) — robustness curves per attack.



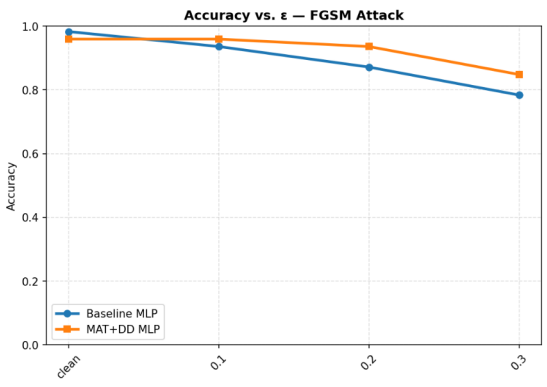
(a) Blocking rate by attack type (adversarial test).

(b) Global pipeline (blocked adv., clean kept, false rejects).

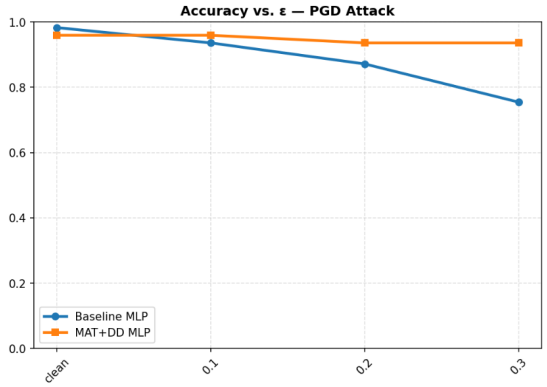
Figure 5.17: Detector effectiveness and end-to-end impact.

Interpretation. The distilled classifier preserves fair clean performance (Acc 0.71) and remains comparatively resilient to C&W (c_0=0.3) while ℓ_∞ attacks still reduce accuracy as ε grows (PGD/BIM at $8/255$ reach ≈0.30–0.32). The detector complements the classifier by rejecting most ℓ_∞ adversarials (99–100% blocking for FGSM/PGD/BIM in this split), which drives the overall *blocked-adversarial* rate to ≈77.6%. However, C&W remains harder to flag (visible dip in the breakdown), and the pipeline incurs ∼15% false rejects on clean inputs, limiting the *accepted clean* rate to ∼71.8%. In short, Variant 2 trades some clean throughput for strong protection against ℓ_∞ attacks, with C&W detection as the main residual weakness.

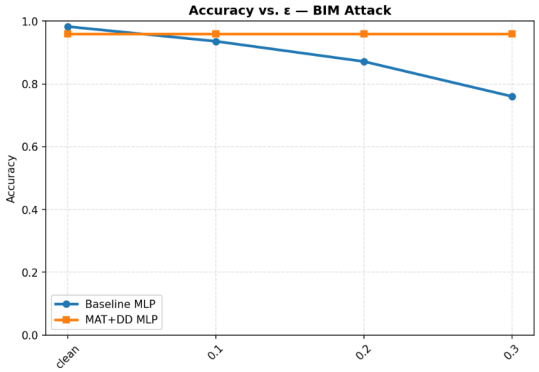
5.6.2 Scenario B — Tabular (WDBC, MLP))



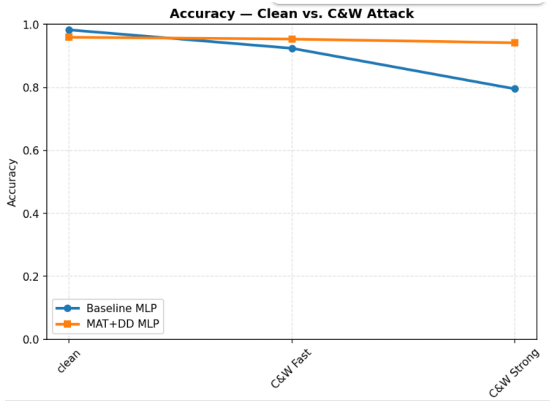
(a) FGSM — Accuracy vs. ε (anchored at *clean*).



(b) PGD — Accuracy vs. ε (anchored at *clean*).

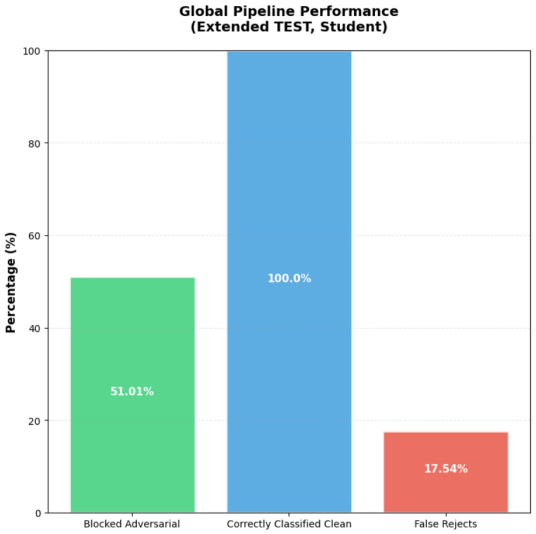


(c) BIM — Accuracy vs. ε (anchored at *clean*).

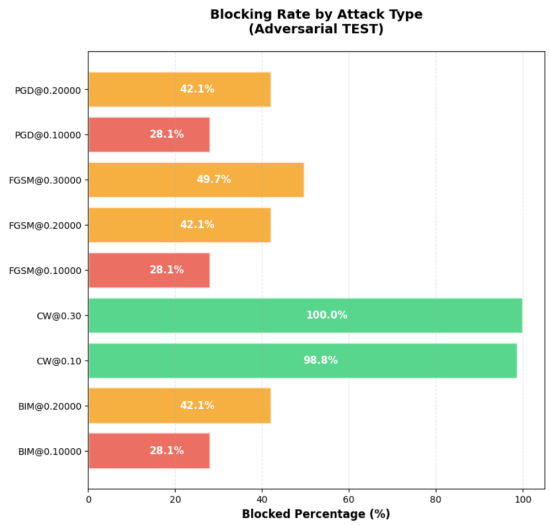


(d) C&W–L2 — Accuracy: clean vs. fast/strong settings.

Figure 5.18: Distilled MLP (no detector) — accuracy per attack.



(a) Global pipeline performance (blocked adversarial, accepted clean, false rejects).



(b) Blocking rate by attack: BIM/FGSM/PGD (28–50%), C&W (99–100%).

Figure 5.19: Detector behavior in the hybrid pipeline (WDBC/MLP).

Interpretation. The distilled MLP remains very strong on clean data (Acc ≈0.96) and sustains high performance under moderate ℓ_∞ budgets (FGSM/PGD/BIM at ε=0.1), with a gradual drop at ε=0.3; C&W (both fast and strong) still yields high accuracy (≈0.94–0.95). The detector complements the classifier but is conservative: it blocks ∼51% of adversarials overall, with high sensitivity to C&W (99–100% blocked) but lower rates for ℓ_∞ perturbations (28–50%). Clean acceptance on the subset of accepted samples is perfect (100%), albeit with ∼17.5% false rejects on the clean stream, highlighting the usual precision–recall trade-off at the detector threshold.

Chapter 6 General Discussion

Overall, our experiments confirm that robustness is strongly dependent on both modality and perturbation budget. The CNN on radiography degrades quickly under small ℓ_∞ perturbations (PGD/BIM stronger than FGSM), whereas the WDBC MLP keeps excellent clean performance (~ 0.98) and only erodes meaningfully at larger ε. Mixed Adversarial Training (MAT) substantially improves resilience to ℓ_∞ attacks at small–moderate budgets, at the cost of a modest clean-accuracy drop; this effect is pronounced in vision and remains beneficial on tabular data. Defensive Distillation, when applied to a robust teacher, yields a compact student that preserves most robustness while maintaining high clean accuracy, making it attractive for deployment. On WDBC, Random Noise Injection (RNI) slightly raises clean accuracy ($\approx 0.983 \rightarrow 0.994$) but does not translate into stronger ℓ_∞ robustness: across FGSM/PGD/BIM, accuracy is consistently below the baseline as ε grows (for example, at ε=0.3, about $0.73/0.68/0.70$ versus $0.78/0.75/0.76$). The C&W–ℓ_2 "fast" setting remains relatively high (~ 0.918), consistent with the idea that small isotropic noise helps little against iterative ℓ_∞ attacks on low-dimensional tabular features; here RNI behaves more like a clean-data regularizer than a robustness booster, and is less effective than MAT or distillation in this scenario. The adversarial detector usefully complements robust training: on images it blocks most PGD/BIM/FGSM examples but is sensitive to thresholding and to C&W; on tabular data the blocking rates are more moderate but consistent with a configuration that favors acceptance of clean inputs. Taken together, the hybrid MAT + Distillation + Detector pipeline offers the best precision–security balance: high clean accuracy, improved robustness at common budgets, and the ability to reject suspicious inputs. Residual vulnerabilities remain under large perturbation budgets or adaptive attackers, but the overall trade-off is the most favorable.

Chapter 7 Contributions and Limitations

Contributions

- Unified multi-modality framework covering imaging (COVID-19 Radiography) and tabular (WDBC) tasks, evaluated under five defense families: Mixed Adversarial Training (MAT), Defensive Distillation, Random Noise Injection (RNI), detector-based rejection, and an end-to-end hybrid pipeline.
- Reproducible protocol with explicit hyperparameters, attack spaces and budgets, and standard metrics (accuracy, precision, recall, F1, ROC–AUC), enabling comparison across scenarios.
- Budget- and norm-aware analysis spanning both ℓ_∞ (FGSM, PGD, BIM) and ℓ_2 (C&W) attacks, highlighting monotonic trends, single-step non-monotonicity, and iteration sensitivity.
- Robust teacher and distilled student configuration showing partial transfer of robustness with favorable size and efficiency for deployment.
- Detector integrated on student embeddings with joint reporting of adversarial block rate, false rejections of clean inputs, and impact on downstream task accuracy.
- Practical hybridization combining MAT, Defensive Distillation, and the detector, yielding the best overall compromise between clean accuracy, robustness, and filtering of suspicious inputs.
- Stronger detector variant (Version 2) with batch normalization and dropout that improves detection at a comparable false-positive rate, strengthening the hybrid pipeline results.
- Clarification of RNI's role on WDBC, where it primarily regularizes clean performance and offers limited gains against iterative ℓ_∞ attacks on low-dimensional tabular features.

Limitations

- Incomplete attack coverage, with limited exploration of adaptive and expectation-over-transformation strategies, as well as comprehensive suites such as AutoAttack.
- Sensitivity to detector threshold calibration and potential distribution shift between calibration and deployment, affecting false rejects and accepts.
- Dependence on preprocessing choices (normalization, clamping) that define the effective attack domain and influence measured robustness.
- Dataset scale and diversity constraints for both imaging and tabular settings, limiting conclusions on generalization to larger, noisier, or more imbalanced datasets.
- Time and cost constraints that restrict hyperparameter sweeps, restarts, and the breadth

of attack grids; robust training (MAT) and iterative attacks (PGD/BIM, strong C&W) are particularly compute intensive.

- Substantial compute requirements for imaging experiments with CNNs, where GPU memory and runtime limitations prevented deeper architecture and parameter explorations.
- Lack of certified guarantees for RNI, which remains susceptible to adaptive attackers aware of the stochastic defense.
- Transferability not yet established across alternative architectures (e.g., ResNet, ViT, transformer-based tabular models) and other data modalities such as text or sensor streams.

Chapter 8 Conclusion and Future Work

conclusion

This work offered a structured, cross-domain view of how AI systems can be stressed and defended, using imaging and tabular settings under a common evaluation lens. We compared several defenses and highlighted the real-world trade-offs between robustness, accuracy, and usability. The key takeaway is that no single mechanism suffices: a coherent combination of hardening, knowledge transfer, and detection delivers the most practical protection. Our hybrid setup exemplifies this systems view and achieves a convincing balance between security and operational performance, while also surfacing deployment considerations such as calibration, acceptable false-reject rates, and compute budget. Beyond headline metrics, the study underscores that robustness must be engineered end-to-end, from data and training curricula to test-time safeguards and monitoring. Vulnerabilities remain under large budgets or adaptive adversaries, but the overall trade-off achieved by the hybrid approach is the most favorable.

Future Work

- Scale up data and models. Move to larger, more diverse datasets and modern architectures; study how robustness behaves at scale and how well results transfer across sites, devices, and populations.
- Optimize compute and cost. Design budget-aware training and evaluation schedules, lightweight inference paths, and deployment-friendly defenses that respect latency, memory, and energy constraints.
- Discover new defenses and hybridization. Explore novel mechanisms (e.g., improved adversarial curricula, test-time adaptation, uncertainty-aware filtering) and systematically compose them with distillation and detection; investigate joint, detector-aware training objectives for stronger hybrids.
- Stronger, more efficient detectors. Develop lean detectors for real-time and edge settings; compare supervised, self-supervised, energy/score-based, and conformal approaches with an emphasis on calibration and low false-reject rates; assess open-set and out-of-distribution handling.
- Broader adversarial evaluation. Include adaptive evaluations that account for defense details, community benchmarks (e.g., AutoAttack), expectation-over-transformation strategies for stochastic defenses, and certified assessments (e.g., randomized smoothing, interval-bound propagation) where feasible.
- Data-centric robustness. Improve curation, preprocessing, and augmentation; address class

imbalance and distribution shift; measure cross-domain and cross-institution generalization and robustness transfer.

- Monitoring and governance. Establish post-deployment monitoring, drift detection, and incident-response playbooks; support reproducible reporting, auditability, and standardized benchmarks to track robustness over time.
- Human-in-the-loop and interpretability. Integrate uncertainty estimation and explanation tools to surface low-confidence or atypical inputs for human review in safety-critical settings.
- Benchmarking and reproducibility. Release standardized protocols, code, and evaluation harnesses to enable fair comparisons and accelerate progress in practical robustness.

Bibliography

Aldahdooh, A., W. Hamidouche, S. A. Fezza, and O. Déforges, "Adversarial Example Detection for DNN Models: A Review and Experimental Comparison," *Artificial Intelligence Review*, 2022, *55* (6), 4403–4462.

Capgemini Research Institute, "97% of organisations hit by GenAI-related security breaches, survey finds," https://www.techmonitor.ai/technology/cybersecurity/97of-organisations-hit-by-gen-ai-related-security-breaches-survey-finds. Accessed: 2025-08-23.

Carlini, Nicholas and David Wagner, "Towards Evaluating the Robustness of Neural Networks," in "IEEE Symposium on Security and Privacy" IEEE 2017.

Chandrasekaran, V., K. Chaudhuri, I. Giacomelli, S. Jha, and S. Yan, "Exploring Connections Between Active Learning and Model Extraction," in "29th USENIX Security Symposium (USENIX Security 20)" 2020, pp. 1309–1326.

Chowdhury, Muhammad E. H., Tawsifur Rahman, Amith Khandakar et al., "Can AI Help in Screening Viral and COVID-19 Pneumonia?," *IEEE Access*, 2020, *8*, 132665–132676.

— , — , — , and — , "COVID-19 Radiography Database," Kaggle 2020. Subset used: COVID vs Normal.

Ezzeddine, F., O. Ayoub, and S. Giordano, "Knowledge Distillation-Based Model Extraction Attack Using GAN-Based Private Counterfactual Explanations," *arXiv preprint arXiv:2404.03348*, 2024.

Geiping, Jonas et al., "Witches' Brew: Industrial Scale Data Poisoning via Gradient Matching," *arXiv preprint arXiv:2009.02276*, 2020.

Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy, "Explaining and Harnessing Adversarial Examples," *arXiv preprint arXiv:1412.6572*, 2014.

Huang, W. Ronny, Jonas Geiping, Liam Fowl, Gavin Taylor, and Tom Goldstein, "Metapoison: Practical General-Purpose Clean-Label Data Poisoning," in "Advances in Neural Information Processing Systems (NeurIPS)," Vol. 33 2020, pp. 12080–12091.

Juuti, Mikko, Sebastian Szyller, Samuel Marchal, and N. Asokan, "PRADA: Protecting Against DNN Model Stealing Attacks," in "2019 IEEE European Symposium on Security and Privacy (EuroS&P)" IEEE 2019, pp. 512–527.

Kariyappa, Santhosh and Moinuddin K. Qureshi, "Defending Against Model Stealing Attacks with Adaptive Misinformation," in "Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)" 2020, pp. 770–778.

Khaddaj, Ahmad et al., "Rethinking Backdoor Attacks," in "International Conference on Machine Learning (ICML)" PMLR 2023, pp. 16216–16236.

Knauer, Jonas, Patrick Rieger, Homa Fereidooni, and Ahmad-Reza Sadeghi, "Phantom: Untargeted Poisoning Attacks on Semi-Supervised Learning," in "Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security" 2024, pp. 615–629.

Koh, Pang Wei and Percy Liang, "Understanding Black-Box Predictions via Influence Functions," in "Proceedings of the 34th International Conference on Machine Learning (ICML)" PMLR 2017, pp. 1885–1894.

Kurakin, Alexey, Ian J. Goodfellow, and Samy Bengio, "Adversarial Examples in the Physical World," *arXiv preprint arXiv:1707.08945*, 2017.

Lee, Tianyu, Ben Edwards, Ian Molloy, and Dong Su, "Defending Against Machine Learning Model Stealing Attacks Using Deceptive Perturbations," *arXiv preprint arXiv:1806.00054*, 2018.

Liang, J., R. Pang, C. Li, and T. Wang, "Model Extraction Attacks Revisited," in "Proceedings of the 19th ACM Asia Conference on Computer and Communications Security" 2024, pp. 1231–1245.

Madry, Aleksander, Aleksandar Makelov, Dimitris Tsipras, and Adrian Vladu, "Towards Deep Learning Models Resistant to Adversarial Attacks," *arXiv preprint arXiv:1706.06083*, 2017.

Muñoz-González, Luis, B. Pfitzner, M. Russo, J. Carnerero-Cano, and E. C. Lupu, "Poisoning Attacks with Generative Adversarial Nets," *arXiv preprint arXiv:1906.07773*, 2019.

Orekondy, Tobias, Bernt Schiele, and Mario Fritz, "Prediction Poisoning: Towards Defenses Against DNN Model Stealing Attacks," *arXiv preprint arXiv:1906.10908*, 2019.

Orhan, Görkem, José Ribeiro, Silas Pohl, and Sofia Costa, "A Comparative Analysis of TRIM and RONI."

Papernot, Nicolas and Patrick McDaniel, "Extending Defensive Distillation," *arXiv preprint arXiv:1705.05264*, 2017.

__ **, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami**, "The Limitations of Deep Learning in Adversarial Settings," in "IEEE European Symposium on Security and Privacy (EuroS&P)" IEEE 2016.

Rahman, Tawsifur, Amith Khandakar, Yazan Qiblawey et al., "Exploring the Effect of Image Enhancement Techniques on COVID-19 Detection Using Chest X-ray Images," *Computers in Biology and Medicine*, 2021, *132*, 104319.

Shafahi, Amirhossein, W. Ronny Huang, Mahyar Najibi, Omid Suciu, Christian Studer, Tudor Dumitras, and Tom Goldstein, "Poison Frogs! Targeted Clean-Label Poisoning Attacks on Neural Networks," in "Advances in Neural Information Processing Systems (NeurIPS)" 2018.

Shahid, A. R., A. Imteaj, P. Y. Wu, D. A. Igoche, and T. Alam, "Label Flipping Data Poisoning Attack Against Wearable Human Activity Recognition System," in "2022 IEEE Symposium Series on Computational Intelligence (SSCI)" IEEE 2022, pp. 908–914.

Shollo, A., B. Doolin, and F. Djellal, "Artificial Intelligence: Definition and Background," in "Artificial Intelligence in Organizations," Springer, 2022, chapter 2.

Szyller, Sebastian, Baris Gecer Atli, Samuel Marchal, and N. Asokan, "DAWN: Dynamic Adversarial Watermarking of Neural Networks," in "Proceedings of the 29th ACM International Conference on Multimedia" 2021, pp. 4417–4425.

Tolpegin, Vahab, Sami Truex, Mehmet Emre Gursoy, and Ling Liu, "Data Poisoning Attacks Against Federated Learning Systems," in "Computer Security – ESORICS 2020: 25th European Symposium on Research in Computer Security" Springer 2020, pp. 480–501.

Truong, Jean-Baptiste, Pratyush Maini, Robert J. Walls, and Nicolas Papernot, "Data-Free Model Extraction," in "Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)" 2021, pp. 4771–4780.

Wang, T. et al., "Deep Learning Model Security: Threats and Defenses," *arXiv preprint arXiv:2412.08969*, 2024.

William, Mangasarian Olvi Street Nick Wolberg and W. Street, "Breast Cancer Wisconsin (Diagnostic)," UCI Machine Learning Repository 1993. DOI: https://doi.org/10.24432/C5DW2B.

Zhao, Kun, Ling Li, Kai Ding, Neil Zhenqiang Gong, Yanjiao Zhao, and Yuxin Dong, "A Survey of Model Extraction Attacks and Defenses in Distributed Computing Environments," *arXiv preprint arXiv:2502.16065*, 2025.

Zhao, Pengfei, Wenqi Zhu, Peng Jiao, Dongdong Gao, and Ou Wu, "Data Poisoning in Deep Learning: A Survey," *arXiv preprint arXiv:2503.22759*, 2025.