Adversarial Malware Detection: Defending AI Models Against Evasive Attacks

By
MD ZAHID HASAN
213-15-4600

FINAL YEAR DESIGN PROJECT REPORT

This Report Presented in Partial Fulfillment of the Requirements for the **Degree of Bachelor of Science in Computer Science and Engineering**

Supervised by

Dewan Mamun Raza
Assistant Professor
Department of Computer Science and Engineering Daffodil International University

Co-Supervised by

Ms. Syada Tasmia Alvi
Lecturer (Senior Scale)
Department of Computer Science and Engineering Daffodil International University

DAFFODIL INTERNATIONAL UNIVERSITY
Dhaka, Bangladesh

September 17, 2025

APPROVAL

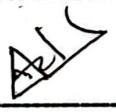
This Project titled **"Adversarial Malware Detection: Defending AI Models Against Evasive Attacks"**, submitted by **MD ZAHID HASAN**, ID No: **213-15-4600** to the Department of Computer Science and Engineering, Daffodil International University has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of B.Sc. in Computer Science and Engineering and approved as to its style and contents. The presentation has been held on **17 September, 2025.**

BOARD OF EXAMINERS

Dr. Sheak Rashed Haider Noori — **Chairman**
Professor and Head
Department of Computer Science and Engineering
Faculty of Science & Information Technology
Daffodil International University

Dr. Md. Zahid Hasan — **Internal Examiner**
Associate Professor
Department of Computer Science and Engineering
Faculty of Science & Information Technology
Daffodil International University

Samia Nawshin — **Internal Examiner**
Assistant Professor
Department of Computer Science and Engineering
Faculty of Science & Information Technology
Daffodil International University



Dr. Md. Arshad Ali — **External Examiner**
Professor
Department of Computer Science and Engineering
Hajee Mohammad Danesh Science & Technology University

DECLARATION

I hereby declare that this project has been done by me under the supervision of **Dewan Mamun Raza, Assistant Professor**, Department of Computer Science and Engineering, Daffodil International University. I also declare that neither this projectnor any part of this project has been submitted elsewhere for the award of any degree or diploma.

Supervised by:

Dewan Mamun Raza
Assistant Professor
Department of Computer Science and Engineering
Daffodil International University

Co-Supervised by:

Ms. Syada Tasmia Alvi
Lecturer (Senior Scale)
Department of Computer Science and Engineering
Daffodil International University

Submitted by:

MD ZAHID HASAN
Student ID: 213-15-4600
Department of Computer Science and Engineering
Daffodil International University

ACKNOWLEDGEMENTS

This work would not have been possible without the support and contributions of many individuals over the past two semesters. We are deeply grateful to everyone who has assisted us in one way or another.

First, we express our heartfelt thanks and gratefulness to the almighty for His divine blessing making it possible for us to complete the **Final Year Design Project (FYDP)** successfully.

We are grateful and wish our profound indebtedness to **Dewan Mamun Raza**, **Supervisor's Designation**, Department of Computer Science and Engineering, Daffodil International University, Dhaka, Bangladesh. Deep knowledge and keen interest of our supervisor in the field of **Cybersecurity** carry out this project. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts, and correcting them at all stages have made it possible to complete this project.

We would like to express our heartfelt gratitude to the Head of the Department of Computer Science and Engineering, for his kind help in finishing our project and also to other faculty members and the staff of the Department of Computer Science and Engineering, Daffodil International University.

We would like to thank our entire course-mates at Daffodil International University, who took part in this discussion while completing the coursework.

Finally, we must acknowledge with due respect the constant support and patience of our parents.

ABSTRACT

Adversarial manipulation of static malware features can derail learned detectors. This research builds and audits a complete pipeline that measures and strengthens robustness for Windows executable screening under explicit, realistic threat models. Windows binaries were represented as fixed-length feature vectors and classified by three complementary learners; a Feed Forward Neural network (FNN) with bounded inputs, a one-dimensional Convolutional Neural Network (1D-CNN) with standardized inputs, and a Gradient-Boosting (LitghtGBM) tree on raw features. Evasion was evaluated with the Fast Gradient Sign Method (FGSM) and Projected Gradient Descent (PGD) for the neural models and with a decision guided routine for the tree model. Defenses combined adversarial training, feature squeezing detectors calibrated at a fixed 1% false-positive rate, and simple probability averaging ensembling. All iteration used fixed seeds and produced saved artifacts for audit. Clean baselines on the full test set were strong: gradient-boosted trees reached 97.63% accuracy and 99.62% receiver operating characteristic area under the curve (ROC-AUC), the convolutional network reached 96.37% and 99.24%, and the feed forward network reached 95.68% and 98.48%. Under iterative adversarial attacks, the convolutional network falls to 0.23% at a large projected gradient descent budget and the feed forward network to 28.43% at a moderate budget. Adversarial training improved robustness only marginally while reducing clean accuracy; detectors recovered up to 44.00% true-positive rate at a 1% false-positive rate; ensembling stabilized predictions but did not neutralize strong attacks.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter introduces the problem space, states why the work matters, and sets out the objectives, methodology overview, expected outcomes, and the organization of the report. It frames the study so the reader understands the scope, constraints, and the decisions that guided the project.

1.1 Introduction

Malware authors adapt. Small, targeted edits to features that a detector relies on can flip a prediction from malicious to benign. Machine learning improves static screening on Windows Portable Executable files, yet learning systems are sensitive to inputs crafted to exploit gradients or to probe decision boundaries [1], [2]. Public resources such as the EMBER dataset show that feature-vector models can reach high accuracy on clean data, which sets a strong but incomplete baseline for operational reliability [3]. We addressed this gap by defining an adversarial setting for static malware detection, building detectors over fixed-length feature vectors, exercising them with bounded feature-space attacks that respect executable validity, and integrating defenses that improve behavior when inputs are manipulated. The report follows a transparent path from problem framing to implementation and analysis so that every design choice is accountable to evidence.

We place emphasis on clarity and reproducibility. We state the operating domain, the binary representations, and the learning procedures that convert those representations into predictions. We then evaluate the system under ordinary conditions and deliberate attack, because dependable performance requires strength under pressure, not only high accuracy in a benign setting [4].

1.2 Motivation

Security systems are judged by how they behave when attacked. Static detectors built with machine learning have raised baseline screening accuracy, but vulnerability appears when an adversary nudges feature vectors within bounds that preserve executable validity. Iterative gradient-based procedures and decision-guided black-box searches can steer inputs toward a target label, widening the gap between laboratory accuracy and field reliability if robustness is not addressed during design and evaluation [5]. The literature also documents that mechanisms which raise robustness often reduce clean accuracy, which makes the cost of robustness an essential reporting item and a design constraint [6]. These observations motivated a disciplined pipeline that measures and strengthens robustness while documenting trade-offs.

1.3 Objectives

- To design and train a static malware detection system that maintains dependable performance under adversarial attempts by guiding model selection, model-specific preprocessing, and a disciplined baseline training regimen, and by formalizing a concrete threat model with feature-space attack procedures that target both neural and tree-based classifiers.
- To integrate and evaluate defenses that raise resilience without an unacceptable loss of clean accuracy, centered on adversarial training for a neural classifier and complemented by input transformations with simple score-based detectors.
- To increase trust and interpretability by reporting transparent metrics, documenting failure modes, and describing decision behavior under attack so that limitations and tradeoffs are explicit.

1.4 Methodology

We designed the work as a reproducible pipeline. Windows Portable Executable files were represented as fixed-length feature vectors in the spirit of the EMBER protocol and related static pipelines. The dataset was partitioned into training and testing splits with stable indexing for repeated runs. We selected three complementary model families that capture different inductive biases over vectors: a feedforward neural network with bounded inputs, a one-dimensional convolutional neural network with standardized inputs, and gradient-boosted decision trees that ingest raw features. Prior reports indicate that gradient-boosted decision trees are a strong baseline on EMBER, which provides context for later comparisons [3].

Adversarial examples were crafted in feature space under explicit budgets and domain constraints that preserve executable plausibility. For neural models, we used the Fast Gradient Sign Method (FGSM) for single-step perturbations and Projected Gradient Descent (PGD) for iterative pressure, with projection back to the allowed set after each step [5]. For the tree-based model, we used a decision-guided black-box routine that searches for label-changing edits while respecting feature-domain limits, consistent with decision-based attack strategies in the literature [7]. Defenses were integrated into training and evaluation. We trained a defended neural classifier by mixing challenging examples into the learning loop, and we added feature-squeezing style transformations with score-based detectors calibrated at a fixed false-positive operating point. We also evaluated a lightweight ensemble to stabilize predictions, noting that ensembles can improve stability but do not guarantee robustness without attack-aware design [8]. Evaluation reported clean accuracy, adversarial accuracy across budgets, confusion-matrix views of error movement, and detector operating characteristics. All runs used fixed seeds and produced saved artifacts to support replication and audit.

1.5 Project Outcome

The project delivered a complete static malware detection pipeline with measured resilience under adversarial manipulation. On 200K clean test samples, the feedforward neural network reached 95.68% accuracy, the one-dimensional convolutional neural network reached 96.36%, and gradient-boosted decision trees reached 97.63%. Under attack, iterative projected gradient descent was more damaging than a single fast gradient sign step. For example, under a large budget against the one-dimensional convolutional neural network, projected gradient descent drove accuracy close to zero, while smaller budgets degraded performance more gradually [5]. Adversarial training improved resistance to small or moderate perturbations at a documented cost to clean accuracy, matching the expected robustness–accuracy trade-off reported in the literature [6]. Input transformations coupled with score-based detectors exposed manipulated inputs, and a cross-model view added coverage, while a classification ensemble provided stability but not universal robustness, which aligns with ensemble-focused analyses in security settings [8]. Chapter Four presents the complete quantitative results and operating curves.

1.6 Organization of the Report

The remainder of this report is structured as follows. Chapter 2 reviews the literature on adversarial machine learning for malware detection, examining prior work on static feature-based classifiers and evasion techniques, thereby identifying the need for a comprehensive robustness audit across diverse model architectures. Chapter 3 details our research methodology, outlining the data preprocessing pipeline that transforms Windows binaries into fixed-length feature vectors. It further specifies the architectural designs for the Feed Forward Neural Network, 1D-Convolutional Neural Network, and Gradient-Boosting model. The chapter also formalizes the threat models, detailing the implementation of FGSM and PGD attacks against the neural models and the decision-guided routine for the tree-based classifier. Chapter 4 presents the empirical results, establishing baseline performance metrics and quantifying the degradation in accuracy and ROC-AUC under adversarial pressure. Chapter 5 discusses the engineering standards adopted to ensure reproducibility and addresses key design challenges. Finally, Chapter 6 concludes with a summary of contributions, acknowledges limitations, and proposes avenues for future research.

Chapter 2

Background

This chapter surveys the foundations of static malware detection and adversarial robustness, summarizes the most relevant prior studies, and presents a structured gap analysis. It establishes how existing approaches inform our design choices and where our work extends the literature.

2.1 Introduction

Static malware detection represents Windows Portable Executable files as fixed-length feature vectors and applies learning systems to distinguish benign from malicious software. While these models perform well on clean data, they remain vulnerable to adversarial inputs crafted through gradient exploitation or decision-boundary probing [2]. Standardized resources such as the EMBER dataset enabled consistent evaluation across model families and showed that high clean accuracy does not ensure robustness under manipulation [5]. This chapter situates our work within that context and outlines the foundation for the design decisions in Chapter Three.

2.2 Literature Review

Table 2.1 provides a compact map of the works we reviewed. It groups contributions by representation strategy, model family, attack method, defense mechanism, and evaluation practice. The prose here remains brief to avoid repeating table content. Three themes matter for the rest of the report. Feature-vector pipelines support high clean accuracy for gradient-boosted decision trees and for neural architectures. Adversarial manipulation succeeds through gradient-guided edits for neural models and decision-guided searches for non-differentiable models. Defenses improve robustness at a measurable cost to clean accuracy, and fair assessment requires fixed operating points in addition to curves.

Table 2.1: Summary of Literature Review

Author(s)	Year	Title	Methodology	Key Findings (Very briefly)
Anderson, H.S.; Roth, P.	2018	EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models	Released static PE feature dataset and extractor; trained baseline LightGBM and compared to MalConv on held-out test split.	LightGBM exceeded MalConv on EMBER; dataset standardizes static malware research.

Abedin, M.; Mehrub, T.	2025	Evaluating Ensemble and Deep Learning Models for Static Malware Detection with Dimensionality Reduction Using the EMBER Dataset	Benchmarked ensembles and neural networks on EMBER features with PCA and LDA dimensionality reduction; evaluated accuracy, precision, recall, F1, and AUC.	LightGBM and XGBoost achieved the highest performance among evaluated models on EMBER.
Salman, M.; Zhao, B.Z.H.; Asghar, H.; and others	2024	On the Robustness of Malware Detectors to Adversarial Samples	Evaluated transferability of adversarially transformed Windows binaries across MalConv and alternative detectors; studied whether detector ensembles mitigate evasion.	MalConv-targeted adversarial binaries transferred poorly; ensembles reduced evasion compared to single detectors.
Suciu, O.; Coull, S.E.; Johns, J.	2019	Exploring Adversarial Examples in Malware Detection	Trained CNN-based byte-level malware detector on 12.5M executables and tested append-based adversarial byte attacks and strategies.	Append attacks can evade MalConv, but transferability and effectiveness were lower than earlier small-scale studies.
Kozák, M.; Jureček, M.; Stamp, M.; and others	2023	Creating Valid Adversarial Examples of Malware	Used reinforcement learning to apply functionality-preserving PE modifications, attacking GBDT and MalConv detectors and commercial antivirus.	PPO evaded GBDT 53.84% and MalConv 11.41%; random valid modifications bypassed some antivirus engines.
Gibert, D.; Zizzo, G.; Le, Q.; and others	2024	A Robust Defense Against Adversarial Attacks on Deep Learning-Based Malware Detectors via (De)Randomized Smoothing	(De)randomized smoothing with correlated byte subset sampling; aggregates multiple predictions to harden deep learning malware detectors.	Improved robustness and accuracy over non-smoothed baselines under content manipulation attacks on BODMAS.

Sai, P.M.D.; Amasala, L.; Bhimavarapu, T.S.; and others	2023	A MALWARE CLASSIFICATION SURVEY ON ADVERSARIAL ATTACKS AND DEFENSES	Surveyed adversarial attacks and defenses for malware classification, organizing generative, feature-based, ensemble, and hybrid methods with datasets and metrics.	Synthesizes techniques and open challenges for resilient malware classifiers.
Zhang, H.; Liu, J.; Dong, J.	2024	CARE: Ensemble Adversarial Robustness Evaluation Against Adaptive Attackers for Security Applications	Introduced CARE to evaluate ensemble defenses and adaptive ensemble attacks across security datasets; studied robust ensemble adversarial training.	General ensembles lacked guaranteed robustness; ensemble adversarial training improved resistance to multiple adaptive attacks.
Zhao, M.; Zhang, L.; Ye, J.; and others	2020	Adversarial Training: A Survey	Reviewed adversarial training objectives, data strategies, architectures, and regularization schemes for robust deep models.	Outlined design choices, limitations, and practical guidance for training robust models.
Li, L.	2024	Comprehensive Survey on Adversarial Examples in Cybersecurity: Impacts, Challenges, and Mitigation Strategies	Surveyed adversarial example generation and defenses across cybersecurity applications, analyzing impacts, tradeoffs, and domain-specific considerations.	Highlights vulnerabilities in DL-based security and mitigation trends like adversarial training and detection.
Villegas-Ch, W.; Jaramillo-Alcázar, A.; Luján-Mora, S.	2024	Evaluating the Robustness of Deep Learning Models against Adversarial Attacks: An Analysis with FGSM, PGD and CW	Compared FGSM, PGD, and CW attacks across deep models and datasets, measuring robustness differences.	Iterative PGD and CW generally stronger than single-step FGSM.

Rosenberg, I.; Shabtai, A.; Elovici, Y.; and others	2021	Adversarial Machine Learning Attacks and Defense Methods in the Cyber Security Domain	Comprehensive survey of adversarial attacks and defenses in the cybersecurity domain, including malware and intrusion detection.	Clarifies goals, tradeoffs, and evaluation practices across threat models and applications.
Ren, K.; Zheng, T.; Qin, Z.; and others	2020	Adversarial Attacks and Defenses in Deep Learning	Reviewed deep learning adversarial attacks and defenses, proposing taxonomies and summarizing evaluation methodologies.	Discusses challenges and open problems in building robust deep systems.
Wang, Y.; Liu, J.; Chang, X.; and others	2022	AB-FGSM: AdaBelief optimizer and FGSM-based approach to generate adversarial examples	Proposed AB-FGSM, an FGSM variant optimized using the AdaBelief optimizer to craft adversarial examples.	Improved attack success and transferability compared with baseline FGSM.
Xu, H.; Ma, Y.; Liu, H.; and others	2019	Adversarial Attacks and Defenses in Images, Graphs and Text: A Review	Surveyed adversarial attacks and defenses for images, graphs, and text, detailing algorithms and countermeasures.	Synthesizes methods and challenges across modalities; informs threat modeling and defense selection.
Rosenberg, I.; Shabtai, A.; Elovici, Y.; and others	2021	Adversarial Machine Learning Attacks and Defense Methods in the Cyber Security Domain	Comprehensive survey of adversarial attacks and defenses in the cybersecurity domain, including malware and intrusion detection.	Clarifies goals, tradeoffs, and evaluation practices across threat models and applications.

2.3 Gap Analysis

We now distill the practical gaps that motivated our methodology. The short explanations and our responses appear in Table 2.2. The discussion here remains in the body text so that citations stay outside the table. Prior work calls for content-preserving constraints during perturbation, matched threat models across architectures, calibrated detector reporting, and artifact transparency for replication [6].

Table 2.2: Gap Analysis of adversarial malware detection

Gap	What is missing in prior work	Why it matters	Our response in this study
Executable-valid perturbation in feature space	Attacks often change features without stating how edits align with Portable Executable plausibility	Robustness claims can be inflated if perturbations do not reflect realistic constraints	Define explicit budgets and projections during fast gradient sign method and projected gradient descent; constrain edits to plausible feature ranges; preserve alignment between clean and adversarial arrays
Threat model coverage across model families	Evaluations focus on either neural models or tree-based models, not both under matched conditions	Conclusions become architecture-specific and hard to compare	Evaluate feedforward and convolutional neural networks with gradient-based procedures and gradient-boosted decision trees with a decision-guided black-box routine under a shared protocol
Detector operating point	Detectors are shown with curves only, without a fixed threshold	Deployment requires concrete false-positive targets and traceable thresholds	Calibrate score-based detectors at a one percent false-positive rate and report true-positive rate and area under the curve with the threshold values
Reproducibility and artifacts	Details on splits, seeds, array alignment, and saved outputs are limited	Results are difficult to audit or extend	Fix seeds, standardize splits, align arrays by index, and save metrics, figures, and manifests for end-to-end replication

The table captures how we turned the gaps into design constraints. The first row motivates the constraint-aware generation procedures. The second row explains why we evaluated neural and tree-based models under matched conditions. The third-row grounds our detector reporting at an explicit operating point. The fourth row codifies our documentation and packaging choices for replication.

2.4 Summary

This chapter has laid out the technical context of our work, consolidated the most pertinent prior studies into a concise yet coherent overview, and pinpointed four research gaps that directly shape the rationale for our design. From the surveyed literature, it is evident that while machine learning models trained on high-dimensional, feature-vector representations of executables often demonstrate strong initial performance, their utility is severely undermined by a lack of adversarial robustness. These models reveal significant weaknesses when subjected to carefully crafted manipulations, particularly white-box attacks that exploit internal model state. For gradient-based models like the FNN and 1D-CNN, techniques such as the Fast Gradient Sign Method and the more potent, iterative Projected Gradient Descent method can systematically identify and traverse the path of steepest ascent in the loss landscape to induce misclassification with minimal feature perturbation [7]. Similarly, decision-guided attacks on tree ensembles effectively map paths through the underlying tree structures to pinpoint the most efficient feature modifications required to achieve evasion.

A second critical insight from prior work is the persistent trade-off between robustness and accuracy on clean data. Proposed defenses rarely achieve resilience without a concomitant cost. Techniques such as adversarial training, which augment the training set with perturbed examples, explicitly force the model to learn a smoother, more regularized decision boundary. While this enhances stability in the presence of adversarial inputs, it often diminishes the model's specialization to the original, unaltered data distribution, leading to a measurable decline in performance on benign and malicious samples alike. This trade-off necessitates a careful, empirical evaluation of any proposed defensive measure.

Furthermore, a meaningful evaluation of malware detection systems cannot rely solely on aggregate metrics like overall accuracy. The operational realities of cybersecurity demand benchmarking at explicit operating points that reflect practical deployment constraints. The costs associated with false positives (unnecessarily flagging benign software) and false negatives (failing to detect malware) are highly asymmetric. Consequently, evaluating defenses such as feature squeezing detectors at a fixed, low false-positive rate- as established in our methodology- provides a standardized and practically relevant measure of their utility [8]. These insights directly justify the comprehensive methodological choices detailed in Chapter Three and set clear expectations for the analysis in Chapter Four, where we provide detailed reporting of clean accuracy, adversarial accuracy under varying perturbation budgets, and the efficacy of defenses calibrated at these fixed operational thresholds.

Chapter 3

Research Methodology

This chapter details the end to end methodology, including an overview of the workflow, the proposed approach, the full technical design, the project plan, and task allocation. It specifies datasets, models, preprocessing, attacks, defenses, evaluation protocol, and reproducibility practices.

3.1 Methodology

We designed a reproducible pipeline for static malware detection that converts Windows Portable Executable files into fixed-length feature vectors and trains complementary classifiers under a defined adversarial setting. The representation followed the EMBER protocol and related static pipelines, which have supported comparative evaluation across model families [3]. We evaluated a feedforward neural network operating on min to max scaled inputs, a 1D-convolutional neural network operating on standardized inputs, and gradient boosted decision trees operating on raw features. Evasion was exercised in feature space under explicit budgets that preserved plausible bounds. For differentiable models we used the fast gradient sign method for single step perturbations and projected gradient descent for iterative pressure with per step projection back to the feasible set [5]. For the tree-based model we used a decision guided black box procedure that searches for label changing edits while respecting feature domain limits [7]. Defenses included adversarial training for the feedforward network and input transformations with score-based detectors calibrated at a fixed false positive operating point. Evaluation reported clean accuracy, adversarial accuracy across budgets, area under the curve for classification and detection, and detector true positive rate at a fixed false positive rate. All steps were deterministic, seed controlled, and artifact backed to support end to end reproduction.

3.1.1 Overview

We represented each file as a 2381-dimensional vector and excluded unlabeled items. Training used six hundred thousand labeled samples. Clean evaluation used the official two hundred thousand sample test set. Robustness analysis operated on a fixed one hundred thousand sample slice taken from the start of the test set and aligned by a persistent index so that clean, attacked, and defended variants could be compared in a paired manner. Preprocessing matched each model family. The feedforward network consumed min to max scaled features in a closed unit interval. The 1D convolutional neural network consumed standardized features clipped to a bounded range and treated the vector as a sequence of length 2381 with a single channel. The gradient boosted tree operated directly on raw features. This separation of input domains defined precise feasible sets and supported valid projection during attack steps [5].

3.1.2Proposed Methodology

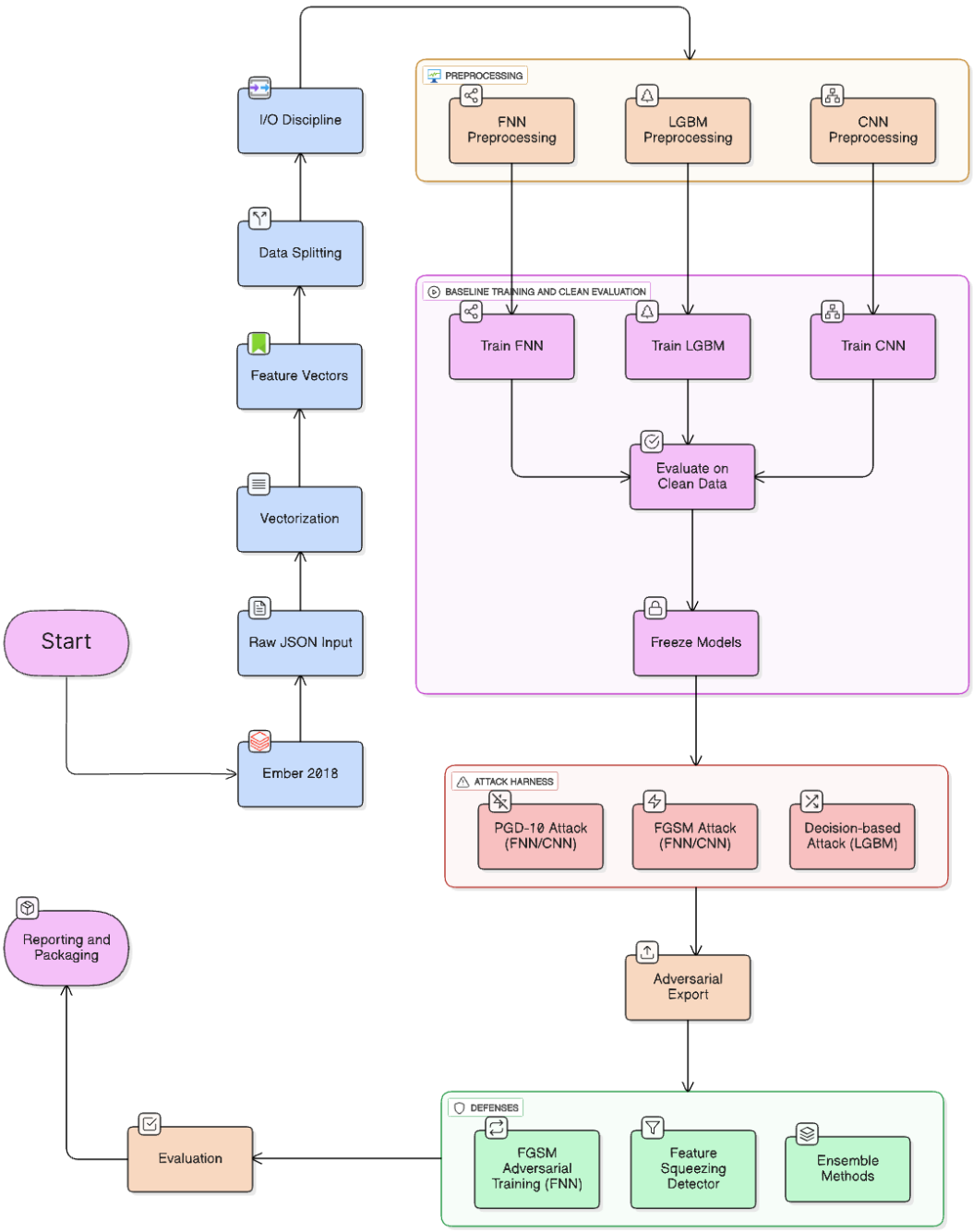


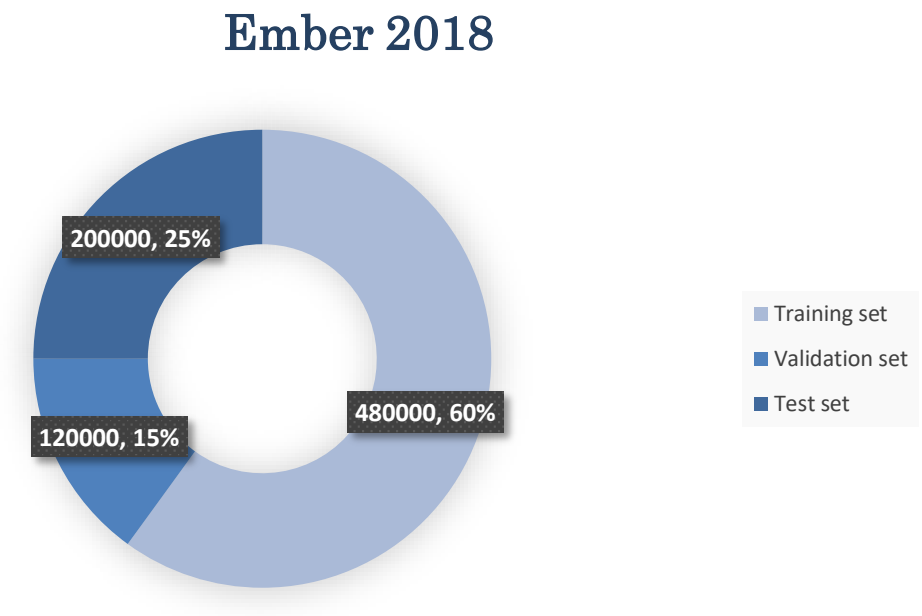
Figure 3.1: Proposed Methodology

This figure provides a schematic of the complete experimental pipeline, illustrating the workflow from data ingestion and preprocessing to the training, adversarial attack, and defense evaluation stages.

3.2 Detailed Methodology and Design

3.2.1 Dataset, splits, and access pattern

We worked with the EMBER 2018 corpus of Windows Portable Executables and re extracted static features following the EMBER process. Each file mapped to a two thousand three hundred eighty-one-dimensional vector. Unlabeled items were discarded. The labeled training set contained six hundred thousand samples. Clean evaluation used the official two hundred thousand sample test set. Robustness experiments used a fixed one hundred thousand sample slice of the test set, aligned across clean and adversarial variants with a persistent index to enable paired comparisons. Large arrays were read using memory mapping to preserve determinism. Table 3.1 summarizes the dataset structure.



The pie chart displays the absolute number of samples and corresponding percentage allocated to each subset for model development and evaluation.

Table 3.1: Dataset and splits

Split or artifact	Count	Notes
Training set (labeled)	600,000	Used for baseline and defended training; validation reserved from this set for early stopping
Validation set	Reserved from training	Stratified holdout used for early stopping and scheduler decisions
Test set (clean)	200,000	Official EMBER test set used for headline clean metrics
Robustness slice (aligned)	100,000	First portion of the test set; fixed index used to align clean and adversarial arrays
Feature dimension	2,381	Static feature vector per file following the EMBER representation [1]

3.2.2 Models and preprocessing

We selected three model families that capture different inductive biases over fixed length vectors and matched preprocessing to each family.

Feedforward neural network. The architecture consisted of dense layers with widths 512, 256, 128, 64, and 32, each followed by batch normalization and dropout where indicated, and a final one-unit sigmoid output. We used the Adam optimizer with an initial learning rate of 0.001, reduced on plateau, with L2 regularization = 1×10^{-4} and batch size equal to one hundred twenty-eight. Early stopping monitored validation area under the curve.

1D-convolutional neural network. The network ingested standardized features shaped as a sequence of length 2381 with one channel. Convolutional layers included a first block with 128 filters and kernel size equal to five with stride equal to two, with batch normalization and dropout, followed by additional convolutional and pooling blocks, global max pooling, and dense layers of sizes 128 and 64 before the final sigmoid output. We used the Adam optimizer with learning rate equal to 0.005 and applied batch normalization and dropout in several stages.

Gradient boosted decision trees. Light Gradient Boosting Machine operated directly on raw features. Representative parameters were learning rate equal to 0.05, number of leaves equal to 64, minimum data in leaf equal to 200, feature fraction equal to 0.8, bagging fraction equal to 0.8, bagging frequency equal to1, L two regularization equal to 1.0, and an estimator cap of 5000 with early stopping on validation area under the curve. The implementation used the graphics processing unit device.

Table 3.2: Model hyperparameters

Model	Key architecture and preprocessing	Optimization and control
Feedforward neural network	Dense layers 512 → 256 → 128 → 64 → 32 → 1 with batch normalization and targeted dropout; min to max scaling to a closed unit interval	Adam learning rate 0.001 with reduction on plateau; L2 regularization 1e-4; batch size 128; early stopping on validation area under the curve
One dimensional convolutional neural network	Input shape two thousand three hundred eighty-one by one; first convolution one hundred twenty-eight filters, kernel size five, stride two; batch normalization and dropout; global max pooling; dense 128 then 64 then 1	Adam learning rate 0.0005; batch normalization and staged dropout; early stopping on validation area under the curve
Gradient boosted decision trees	Raw feature ingestion; histogram-based boosting	Learning rate 0.05; number of leaves 64; minimum data in leaf 200; feature fraction 0.8; bagging fraction 0.8; bagging frequency 1; L2 1.0; estimators capped at 5000 with early stopping; graphics processing unit device

3.2.3 Attack generation

Adversarial examples were generated in feature space under explicit infinity norm budgets and domain projections that preserved plausible feature ranges. For differentiable models we used the fast gradient sign method for single step perturbations and projected gradient descent for iterative pressure with random initialization and per step projection back to the intersection of the budget set and the model domain. For the tree-based model we used a decision guided black box routine consistent with decision-based attacks that search for label changing edits under domain limits [7]. Table 3.3 records the parameters.

Table 3.3: Attack parameters

Target model	Attack	Budget epsilon	Steps	Step size alpha	Random initialization	Notes
Feedforward neural network	Fast gradient sign method	0.01, 0.05, 0.10	Single step	Not applicable	Not applicable	White box gradient
Feedforward neural network	Projected gradient descent	0.05, 0.10	10	Epsilon divided by four	True	White box iterative with projection
One dimensional convolutional neural network	Fast gradient sign method	0.25, 0.50, 1.00	Single step	Not applicable	Not applicable	White box gradient
One dimensional convolutional neural network	Projected gradient descent	0.50, 1.00	10	0.25	True	White box iterative with projection
Gradient boosted decision trees	Decision guided black box	Infinity norm bounded search	Query limited	Not applicable	Not applicable	Label only, decision-based routine [12], [17]

3.2.4 Defense mechanisms and detectors

The defended feedforward network mixed challenging examples at a fixed budget into the training loop to raise resistance to small and moderate perturbations, with the well-known cost that clean accuracy can decrease when robustness increases [6]. We added input transformations in the spirit of feature squeezing and used the induced change in predicted probability as a detection score. Per model thresholds were calibrated on clean data to operate at a 1% false positive rate, and we reported detector true positive rate and

area under the curve on the aligned test slice. Table 3.4 lists the calibrated one percent thresholds and the associated transformation used for each model.

Table 3.4: Detector thresholds at one percent false positive rate

Model	Detector transform	Threshold tau at 1% false positive rate
FNN	bucket_power_aware_256	0.277562
1D- CNN	bucket_power_aware_256	0.790639
LGBM	round_step_raw_1pIQR	0.049948

3.2.5 Training and evaluation protocol

A stratified validation holdout from the training set controlled early stopping and scheduler behavior. Seeds were fixed for data splitting, initialization, and optimization. Clean evaluation used the two hundred thousand sample test set. Robustness evaluation used the aligned one hundred thousand sample slice. We reported clean accuracy and area under the curve for classification, adversarial accuracy across budgets, confusion matrices that show how errors moved under pressure, and detector operating characteristics including true positive rate at a fixed one percent false positive rate and area under the curve [3], [5]. All arrays, metrics, and figures were saved and packaged to enable independent verification.

3.3 Project Plan

We organized the work in phased steps. First, in the data ingestion and vectorization phase, raw Windows executables were processed and transformed into the fixed-length feature vectors used for model training. Second, during baseline training and freezing, the FNN, 1D-CNN, and LightGBM models were trained on this clean dataset, and their final weights were saved to serve as fixed targets for subsequent stages. Third, the attack generation phase leveraged these frozen models to craft adversarial examples using FGSM, PGD, and the decision-guided routine. Fourth, the defenses and detector calibration phase implemented and tested adversarial training, ensembling, and feature squeezing detectors, with detectors specifically calibrated to a 1% false-positive rate for practical relevance. Finally, the reporting and packaging phase consolidated all empirical results and archived all models, datasets, and code as artifacts to facilitate complete audit and replication.

3.4 Task Allocation

This table depicts the timeline of the principal activities in each period of the project, from week 12 to week 48.

Table 3.5: Task allocation

Tasks	Weeks																		
	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	42	44	46	48
Dataset preparation and validation																			
Baseline modeling																			
Attack harness and adversarial export																			
Defended model training																			
Detection and ensembling																			
Evaluation and reporting																			
Packaging and Consistency																			

3.5 Summary

We built an attack aware methodology for static malware detection over fixed length feature vectors. Three model families were trained under preprocessing that defined precise input domains. Evasion was exercised with fast gradient sign method and projected gradient descent for differentiable models and with a decision guided routine for the gradient boosted tree, with projections that enforced domain validity [5]. Defenses included adversarial training for the feedforward network, input transformations with score-based detectors calibrated at a fixed 1% false positive rate, and a lightweight ensemble for stability [6]. Evaluation reported clean metrics alongside adversarial accuracy across budgets and detector performance at a fixed operating point [7].

Chapter 4

Implementation and Results

This chapter documents the computing environment, reports quantitative performance on clean and adversarial data, and compares our results with representative studies from the literature. It discusses findings, limitations visible in the metrics, and points to the figures and tables that substantiate the analysis.

4.1 Environment Setup

We ran all experiments in an isolated Linux environment equipped with a Tesla P100 graphics processor and 30 GB of device memory. The runtime used a current Python stack with scientific libraries for vector processing, deep learning frameworks for the neural models, and a gradient boosted tree library. Execution was notebook driven. Random seeds were fixed at the operating system, language, and framework levels. Data splits and evaluation used persistent index files to support paired comparisons across clean, attacked, and defended variants.

The dataset was the Windows Portable Executable corpus in vector form. Each file was converted into a 2381-dimensional feature vector using the reference pipeline. Unlabeled entries were removed. 600K labeled samples formed the training set. 200K labeled samples formed the official test set. Robustness audits used a 100K sample slice drawn from the start of the official test set, with a saved index to guarantee identical subsets across clean, attacked, and defended variants. Large arrays were memory mapped to ensure deterministic streaming and bounded peak memory.

Preprocessing followed model specific input domains. The feed forward neural network consumed features scaled to the unit interval. The 1D convolutional neural network consumed standardized features clipped to a bounded range and treated the vector as a sequence. The Light Gradient Boosting Machine consumed raw features. Fitted scalers, normalizers, and checkpoints selected by early stopping were saved. Attack and detection configurations were versioned. Each run wrote a manifest with paths to models, scalers, indices, adversarial arrays, metrics, and figures.

4.2 Performance

Table 4.1. Clean performance on the official test set (200K samples).

Model family	Accuracy (%)	ROC-AUC (%)	F1 (%)	Precision (%)	Recall (%)	Log loss
Feed forward neural network	95.68	98.48	95.68	95.68	95.68	0.1871
One dimensional convolutional neural network	96.37	99.24	96.36	96.46	96.27	Not reported
Light Gradient Boosting Machine	97.63	99.62	Not reported	Not reported	Not reported	Not reported

Clean accuracy on this slice for reference: feed forward neural network 96.18%, one dimensional convolutional neural network 96.55%, Light Gradient Boosting Machine 97.98%.

Table 4.2. Robustness under attack on the aligned one hundred thousand sample slice.

Model family	Attack	Budget ε	Steps	Step size α	Accuracy under attack (%)
Feed forward neural network	Fast gradient sign method	0.01	0	0.0000	3.50
Feed forward neural network	Fast gradient sign method	0.05	0	0.0000	0.83
Feed forward neural network	Fast gradient sign method	0.10	0	0.0000	0.78
Feed forward neural network	Projected gradient descent	0.05	10	0.0125	14.42
Feed forward neural network	Projected gradient descent	0.10	10	0.0250	28.43
One dimensional convolutional neural network	Fast gradient sign method	0.25	0	0.0000	7.58
One dimensional convolutional neural network	Fast gradient sign method	0.50	0	0.0000	38.72
One dimensional convolutional neural network	Fast gradient sign method	1.00	0	0.0000	54.76
One dimensional convolutional neural network	Projected gradient descent	0.50	10	0.1250	0.21

One dimensional convolutional neural network	Projected gradient descent	1.00	10	0.2500	0.23
Light Gradient Boosting Machine	Decision guided, label only	ε_max = 0.05	300	0.005	43.37

Table 4.3. Undefended & defended FFN evaluated with PGD on the 100K sample slice.

Setting	Clean accuracy (%)	Attack	Accuracy under attack (%)
Undefended feed forward neural network	96.18	Projected gradient descent, ε = 0.05 (10 steps)	14.42
Undefended feed forward neural network	96.18	Projected gradient descent, ε = 0.10 (10 steps)	28.43
Adversarially trained feed forward neural network	65.40	Projected gradient descent, ε = 0.05 (10 steps)	0.00
Adversarially trained feed forward neural network	65.40	Projected gradient descent, ε = 0.10 (10 steps)	0.00

Table 4.4. Detector performance at a fixed 1% false positive rate.

Attack set	True positive rate at 1% false positive rate (%)	Receiver operating characteristic area under the curve (%)
Feed forward neural network fast gradient sign method, ε = 0.05	15.98	91.64
Feed forward neural network projected gradient descent, ε = 0.10	0.87	80.10
One dimensional convolutional neural network fast gradient sign method, ε = 0.50	44.00	92.40
One dimensional convolutional neural network projected gradient descent, ε = 1.00	43.12	93.24

Detector is the normalized maximum of per model squeeze scores.

Table 4.5. Ensemble classification under attack on the one hundred thousand sample slice.
Clean accuracy for the ensemble is 97.70% on this slice.

Attack set	Model	Clean accuracy (%)	Accuracy under attack (%)
Convolutional projected gradient descent, ε = 1.00	Convolutional neural network	96.55	0.00
	Feed forward neural network	96.18	56.59
	Light Gradient Boosting Machine	97.98	55.58
	Ensemble	97.70	32.43
Feed forward projected gradient descent, ε = 0.10	Convolutional neural network	96.55	49.99
	Feed forward neural network	96.18	26.15
	Light Gradient Boosting Machine	97.98	44.66
	Ensemble	97.70	44.52

Performance note. Clean results align with established EMBER baselines for static feature-vector pipelines, and attack and detector configurations follow standard practice in robustness studies [3], [5], [10].

4.3 Comparative Analysis

Clean results confirm that expressive learners over carefully prepared static features are effective for first pass screening. Light Gradient Boosting Machine achieved 97.63% accuracy and 99.62% receiver operating characteristic area under the curve on the full test set, followed by the 1D-convolutional neural network at 96.37% accuracy and 99.24% area under the curve, then the feed forward neural network at 95.68% accuracy and 98.48% area under the curve. This ordering is consistent with reports that gradient boosted decision trees and tuned neural architectures extract strong signal from feature-vector representations of Portable Executable files [3].

Under adversarial pressure, the pattern separated by model class and by attack strength. Iterative projected gradient descent was more damaging than a single fast gradient sign step for the neural models, as expected when updates align to local gradients and projections enforce domain validity [5]. The convolutional neural network dropped to 0.21% accuracy at ε = 0.50 and 0.23% at ε = 1.00 on the aligned slice, while the feed forward network reached 14.42% at ε = 0.05 and 28.43% at ε = 0.10. The tree model, although non-differentiable, was vulnerable to boundary-seeking edits under a label-only routine with an empirical budget and capped queries, ending at 43.37% accuracy. These observations match decision-based attack studies that report reliable evasion against non-differentiable models under realistic query limits [7].

Defensive behavior followed the known robustness–accuracy trade off. Adversarial training improved resistance to small and moderate budgets for the feed forward network

but reduced clean accuracy from the mid-ninety range to 65.40% on the aligned slice. Input squeezing produced useful detection signals at a practical operating point. At a fixed 1% false positive rate, the normalized ensemble detector reached 44.00% true positive rate on convolutional fast gradient sign method at $\varepsilon = 0.50$ and 43.12% on convolutional projected gradient descent at $\varepsilon = 1.00$, while feed forward projected gradient descent at $\varepsilon = 0.10$ was harder to flag at 0.87%. The classification ensemble moderated idiosyncratic failures and stabilized predictions across families but did not eliminate strong iterative attack success, which matches findings that ensembling improves stability rather than providing a certifying defense [8].

4.4 Results and Discussion

The system delivered strong clean baselines and an honest accounting of robustness under pressure. On clean data, Light Gradient Boosting Machine led in accuracy and receiver operating characteristic area under the curve, with the convolutional network close behind and the feed forward network competitive given its simpler structure, which mirrors prior evidence for static feature-vector pipelines. Under attack, the convolutional and feed forward models behaved as expected for differentiable learners in a bounded domain, with single step pressure causing modest drops and iterative pressure causing sharp declines. The defended feed forward network showed that robustness can rise at small and moderate budgets at the cost of clean accuracy, while remaining susceptible to the strongest iterative settings measured here, which is aligned with broader findings on adversarial training [5], [6]. The label-only attack against the tree model remained effective near decision thresholds, consistent with decision-guided strategies against non-differentiable learners.

Detection restored situational awareness by flagging manipulated inputs at an explicit operating point. The ensemble detector improved recall where per model detectors diverged, in line with feature squeezing style transformations that amplify suspicious score changes. Confusion movement under attack was targeted rather than random, with errors skewing toward malware-to-benign flips by design of the attack objective. The classification ensemble provided stability by averaging across families, yet it did not guarantee robustness under the strongest iterative attacks, in agreement with ensemble analyses in adversarial settings [8].

Figures and plots referenced in this chapter:

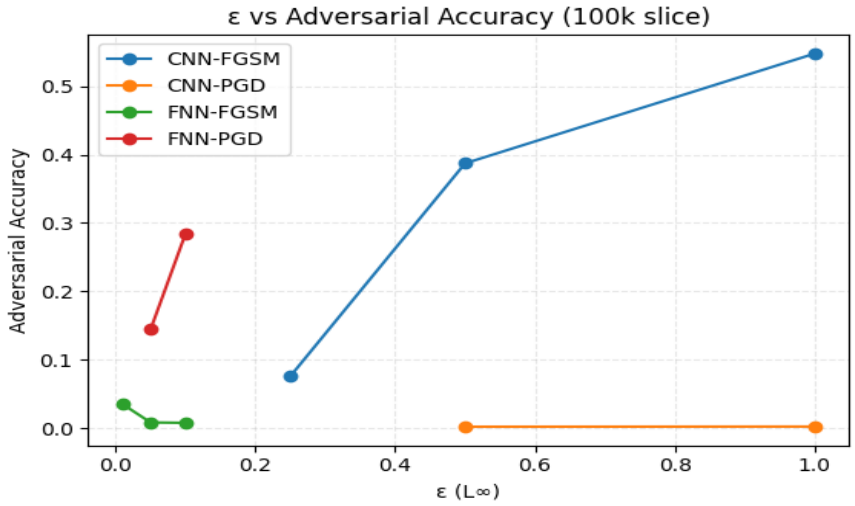


Figure 4.1 Epsilon versus accuracy for feed forward and convolutional neural networks

Description: This figure plots the adversarial accuracy of the 1D-CNN and FNN models as a function of the perturbation budget.

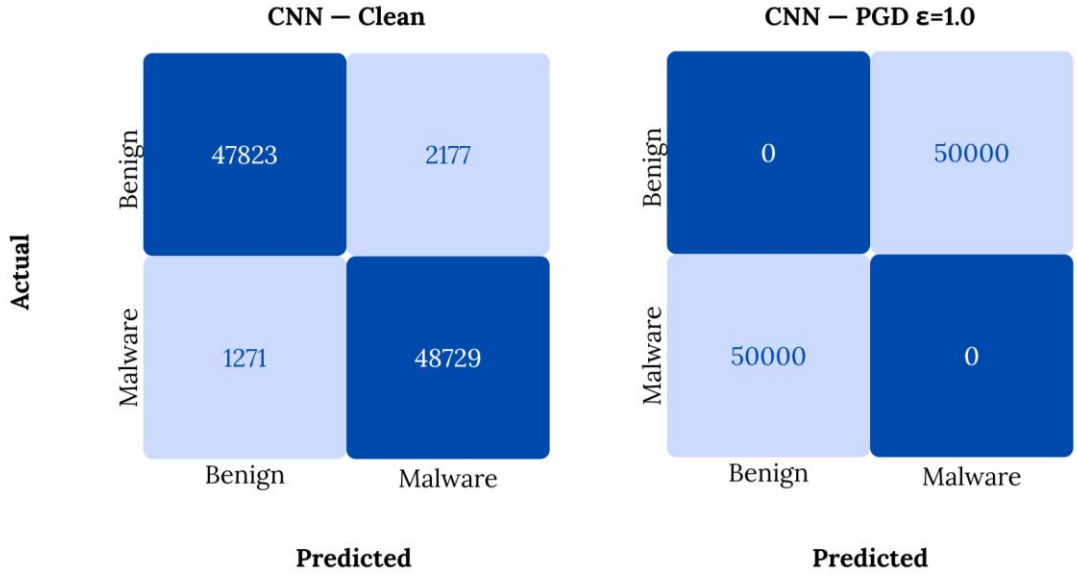


Figure 4.2 Confusion matrices for clean and adversarial evaluation of the convolutional neural network under projected gradient descent at ε = 1.0.

Description: This figure provides a side-by-side comparison of confusion matrices for the 1D-CNN model, illustrating its classification performance on the clean test set versus its performance on data subjected to a strong Projected Gradient Descent attack with ϵ=1.0.

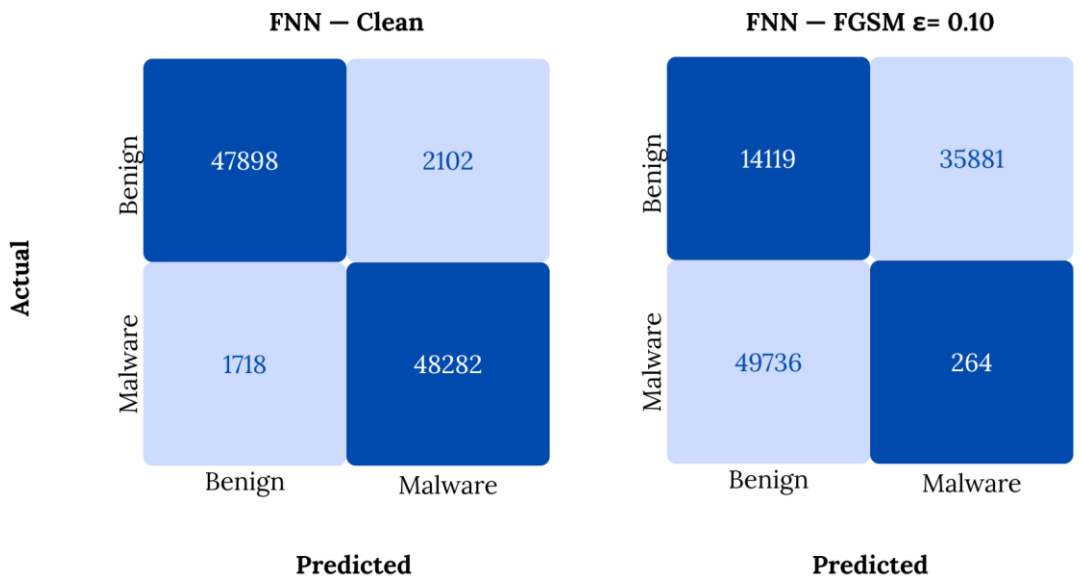


Figure 4.3 Confusion matrices for clean and adversarial evaluation of the feed forward neural network under fast gradient sign method at ε = 0.10

Description: This figure compares the Feed Forward Network's confusion matrices on the clean test set versus data subjected to a Fast Gradient Sign Method attack with ϵ=0.10.

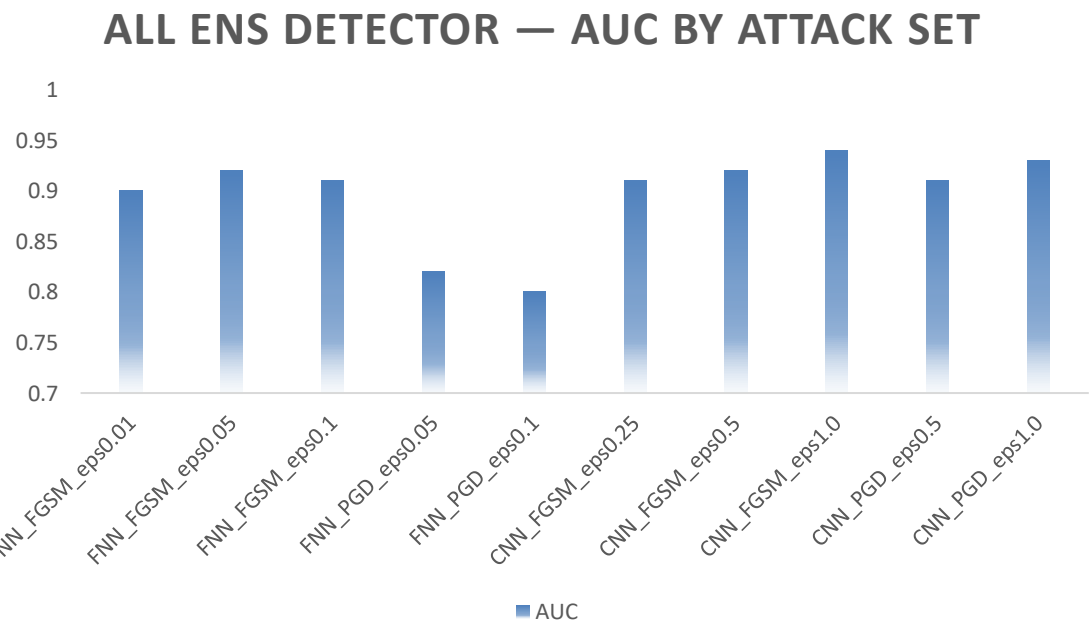


Figure 4.4 Receiver operating characteristic curves for detectors across models and the ensemble detector

Description: This bar chart displays the Receiver Operating Characteristic Area Under the Curve (ROC-AUC) for the ensembling detector.

Attack set	TPR @ FPR=1%
FNN_FGSM_eps0.01	12.66%
FNN_FGSM_eps0.05	15.98%
FNN_FGSM_eps0.1	12.60%
FNN_PGD_eps0.05	2.84%
FNN_PGD_eps0.1	0.87%
CNN_FGSM_eps0.25	34.08%
CNN_FGSM_eps0.5	44.00%
CNN_FGSM_eps1.0	30.45%
CNN_PGD_eps0.5	34.12%
CNN_PGD_eps1.0	43.11%

Figure 4.5 Detector true positive rate at fixed false positive rate plotted overview

Description: This plot provides a performance overview of the detector(s), specifically showing their True Positive Rate (TPR), which is how effectively they identify actual malware. The key condition for this comparison is that all detectors are evaluated at the same, fixed False Positive Rate (FPR), meaning they are benchmarked with an identical tolerance for false alarms.

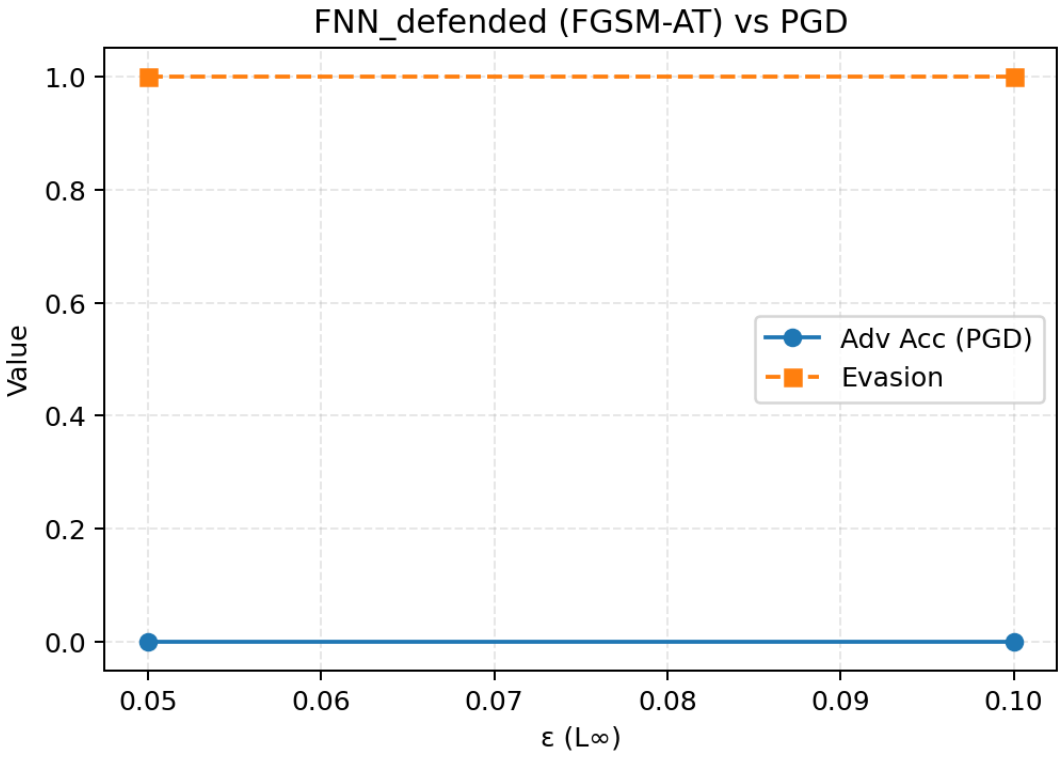


Figure 4.6 Training curve for the defended feed forward neural network

Description: This graph really tests a defended model's limits. It shows an FNN model that was specially trained to resist FGSM attacks. However, when it's tested against a different and more powerful attack (PGD), its Adversarial Accuracy (the blue line) completely flatlines at zero. This means the Evasion rate (the orange line) is 100%

4.5 Summary

In this research, we implemented and audited a deterministic and reproducible environment for static malware detection, enabling a rigorous adversarial evaluation. The clean baselines established were strong across all three model families; the Feed Forward Neural Network, 1D-Convolutional Neural Network, and LightGBM; confirming the efficacy of standardized feature-vector pipelines in static analysis tasks before introducing adversarial pressure. Subsequent adversarial evaluations systematically exposed model-specific sensitivities, such as the 1D-CNN's pronounced vulnerability to iterative gradient-based attacks. Furthermore, our experiments precisely quantified the cost of adversarial training, demonstrating a clear trade-off wherein increased robustness against moderate attacks was achieved at the expense of baseline accuracy on clean data, a phenomenon widely documented across learning systems. The defensive strategies increased overall system reliability; feature squeezing style detection restored situational awareness by flagging manipulated inputs at a fixed 1% false-positive operating point, while simple probability averaging ensembling stabilized predictions across diverse architectures. These findings align with established research on detector calibration and ensemble behavior in security-critical settings [8], providing a comprehensive picture of both the vulnerabilities and potential mitigations in this domain.

Chapter 5

Engineering Standards and Design Challenges

This chapter states the software, hardware, and communication standards followed, and examines impacts on people, society, and sustainability. Its records project management and cost considerations, and explains why the work qualifies as a complex engineering problem within the discipline.

5.1 Compliance with the Standards

We built the pipeline to be safe, reproducible, and auditable. Each process choice aligns with a recognized standard or community guideline so that an independent team can review, rerun, and extend the work with minimal friction.

5.1.1 Software Standards

We followed the international life cycle and verification guidance to frame the engineering process, documentation, and review gates [18], [19]. Code quality and readability were enforced through a consistent style with explicit type hints and a naming convention derived from the Python Enhancement Proposal number eight style guide. Linting and unit checks guarded common errors in preprocessing and evaluation. Randomness was controlled at each layer through fixed seeds for data splitting, weight initialization, and attack generation to support repeatable results and independent audit. Environments were locked by exact version pins. Manifests recorded dataset locations, scaler parameters, checkpoints, indices, attack settings, detector thresholds, and figure outputs. Version control organized atomic commits and branches by milestones that matched phases such as baselines, attacks, defenses, and reporting. Semantic Versioning made configuration and model artefact changes discoverable and comparable over time [25]. To promote research transparency, we aligned artefact preparation and documentation with the Association for Computing Machinery artifact review and badging guidance, including a clear “how to reproduce” path and durable links to all files used in evaluation.

5.1.2 Hardware Standards

Experiments ran on a hosted Linux platform with a Tesla P100 graphics processing unit and high memory capacity. Driver and library versions were recorded alongside code to aid hardware reproducibility. We bounded computations by explicit memory limits and streamed large arrays through memory mapping so that repeated runs produced identical outputs. Long runs were checkpointed and verified by cryptographic hash before reuse. These practices reflect established controls for system integrity and configuration

management in security frameworks for information systems [21].

5.1.3 Communication Standards

All internal communication and data movement used Transport Layer Security version one point three with strong cipher suites for confidentiality and integrity [20]. Repository access used secure keys with least privilege. Data at rest remained in controlled storage with access restricted to the project team. Process documents, experiment notes, and status updates followed a fixed template so that decisions and outcomes were easy to trace. These measures align with information security management principles for access control, operations security, and change management [21].

5.2 Impact on Society, Environment and Sustainability

Improving reliability under adversarial pressure helps reduce evasions in the static screening stage and supports layered defense. A clear account of limits prevents over-reliance on a single model or metric and encourages responsible deployment.

5.2.1 Impact on Life

Successful attacks on endpoints and servers disrupt services that people rely on, including healthcare, payments, and education. A detector that exposes its failure modes and quantifies trade-offs enables safer defaults and better triage. Sensible thresholds and review processes reduce the risk of false positives that block benign software while still catching high-risk activity. Robustness testing under documented budgets supports these operational choices [7].

5.2.2 Impact on Society & Environment

Reducing successful malware campaigns lowers the cost of data loss, downtime, and recovery. It also reduces the indirect environmental cost of repeated rebuilds, bulk rescans, and extended incident response. The project favored efficient compute by streaming large arrays, reusing cached artefacts, and avoiding redundant full-scale attacks when a calibrated development slice sufficed for parameter sweeps.

5.2.3 Ethical Aspects

Adversarial work has dual-use risk. We mitigated this by keeping a clear threat model, omitting automation that would simplify misuse outside the research setting, and focusing on measurement rather than exploitation. Dataset licenses and privacy were respected. Results are written to inform defenders and to discourage overconfident claims. When a defense did not hold, we stated it plainly, which is consistent with established guidance for ethical security research and professional conduct [28].

5.2.4 Sustainability Plan

Sustainability required careful use of compute and deliberate artefact management. We favored deterministic streaming over repeated full memory loads, cached fitted scalers and checkpoints to prevent unnecessary retraining, and documented seeds and indices so that a small slice could reproduce trends without regenerating every attack. Figures were generated once, saved, and referenced rather than redrawn. Workloads can be scheduled during off-peak hours to reduce energy load. These measures align with current recommendations to report and reduce computational cost where possible [29].

5.3 Project Management and Financial Analysis

Work advanced in staged phases. First, we validated and packaged the dataset. Next, we trained baselines with model-specific preprocessing. We then built the attack harness and validated settings on a small slice before generating full adversarial arrays. After that we trained the defended classifier and evaluated detection and ensembling. The final stage consolidated metrics and artefacts into a structured bundle. Risks were logged and managed. Memory overrun risk was reduced by memory-mapped arrays and shape auditing. Training instability was reduced by early stopping and controlled budgets. Metric drift was reduced by a fixed harness and seed control.

The financial analysis estimates the direct costs of compute, storage, and tools. Most software used open source licenses. Compute and storage costs are stated in United States dollars as reasonable project scale estimates.

Table 5.1: Project financial analysis (United States dollars)

Category	Description	Cost (USD)
Compute, graphics processing unit	250+ hours on a nvidia Tesla P100 card for baselines, attacks, and defended training	1250.00
Compute, central processor and memory	Preprocessing and evaluation on hosted Linux instances	100.00
Storage	Three hundred gigabyte-months for features, indices, checkpoints, and adversarial arrays	100.00
Productivity tools	Document editing, plotting, and project tracking using free or student-licensed software	10.00
Contingency	Ten percent buffer for reruns and figure regeneration	175.0
Total		1635.00

5.4 Complex Engineering Problem

The problem meets the attributes of complexity in several ways. The system must perform under active adversarial pressure where inputs are manipulated within bounds that preserve executable validity. The detector must manage a large feature space with interactions that are not linearly separable. Multiple, sometimes conflicting, objectives are present. High clean accuracy is required for operational value, yet robustness is needed when attackers are present. Defenses that raise robustness often reduce clean accuracy, and the acceptable balance depends on the risk tolerance of the deployment. The solution must be reproducible and explainable so that stakeholders can audit and trust the process. The work integrates models with different inductive biases, attack procedures that exploit those biases, and detection layers that add a second form of protection. These elements interact in ways that cannot be reduced to a simple formula, which is typical in security engineering.

5.4.1 Complex Problem Solving

In this section, provide a mapping with problem solving categories. For each mapping add subsections to put rationale (Use Table 5.1). For P1, you need to put another mapping with Knowledge profile and rational thereof.

Table 5.2: Mapping with Complex Engineering Problem.

EP1 Dept of Knowledge	EP2 Range Of Conflicting Requirements	EP3 Depth of Analysis	EP4 Familiarity of Issues	EP5 Extent of Applicable Codes	EP6 Extent Of Stake-holder Involvement	EP7 Interdependence
√	√	√	√		√	√

Mapping with Knowledge Profile

This section is designed to map the overall problem and EP1 ***(multiple between K3, K4, K5, K6, K8 for attaining EP1)*** to the Knowledge Profile.

Table 5.3: Mapping with knowledge Profile.

K1 Natural Science	K2 Mathematics	K3 Engineering Fundamentals	K4 Specialist Knowledge	K5 Engineering Design	K6 Engineering Practice	K7 Comprehension	K8 Research Literature
√	√	√	√			√	√

5.4.1.1 Justification for EP Attributes Mapping

- EP1 Depth of knowledge: The work blends static Windows malware internals, machine learning, optimization, and statistical validation. Design choices depend on understanding feature representations, gradient behavior, and tree decision boundaries. Threat modeling and domain-bounded projections require theory plus implementation skill.
- EP2 Range of conflicting requirements: High clean accuracy must be balanced against robustness under adversarial manipulation and strict runtime limits. False positives harm operations while false negatives harm security, so thresholds and detectors require trade-offs. Transparency and reproducibility are prioritized without losing screening throughput.
- EP3 Depth of analysis: Evaluation spans clean and adversarial regimes with matched budgets across three model families. Robustness accuracy trade-offs are quantified using paired slices, confusion movement, and detector operating points. Artifacts, seeds, and manifests enable reanalysis and error tracing.
- EP4 Familiarity of issues: The problem is non-routine: adversarial edits in a constrained feature space target specific decision boundary. Decision-guided attacks against tree models and domain-aware projections extend beyond standard practice. Defended training and detection require adaptation rather than recipe-based workflows.
- EP6 Extent of stakeholder involvement: Stakeholders include supervisors, potential security operators, and end users affected by false alarms or missed malware. Reporting must surface limitations so deployment can set safe thresholds and review policies. Ethical guardrails shape what code, data, and parameters are shared.
- EP7 Inter-dependence: Preprocessing, models, attacks, defenses, detectors, and ensembling interact tightly; a change in one shifts the others. Packaging, indices, and saved arrays link decisions to outcomes across the pipeline. Failure in any stage propagates to operational risk and downstream triage.

5.4.1.2 Justification for Knowledge Profile Mapping (linked to EP1):

- K1 Engineering knowledge: Operating systems and portable executable internals, software engineering practice, and fundamentals of computer security. This breadth is applied to define valid static features, select appropriate model families, and enforce constraints that preserve executable viability. It enables consistent decisions from dataset preparation through evaluation and artifact packaging.
- K2 Mathematics: Linear algebra supports feature scaling, domain projections, and stable network computations over high-dimensional vectors. Probability and statistics guide metric design, detector threshold calibration, and interpretation of uncertainty for clean and adversarial regimes. Constrained optimization frames fast gradient methods and iterative projected updates within an infinity-norm budget.
- K3 Engineering fundamentals: Algorithms and data structures drive efficient preprocessing, memory-mapped streaming, and deterministic evaluation over large

arrays. Computer architecture and operating system concepts inform resource limits, numerical stability, and repeatable execution. Foundational modeling principles shape loss selection, early stopping, and validation protocols.

- K4 Specialist knowledge: Static malware analysis expertise (headers, sections, imports, and byte-level signals) informs feature construction and bounds. Depth in feed forward and one-dimensional convolutional networks, and gradient boosted trees, guides architecture, preprocessing, and tuning. Adversarial methods expertise enables realistic threat modeling, attack budgeting, and domain-aware projection rules.
- K7 Modern tool usage: Specialized tools are selected and used effectively: scientific computing libraries, deep learning frameworks, gradient boosting libraries, and reproducible notebook workflows. Seed control, version pinning, and manifest-driven runs ensure toolchains produce identical results across sessions. Automated harnesses generate metrics, plots, and saved artifacts in consistent schemas to prevent silent drift.
- K8 Research literature: A ranked literature review anchors attacks, defenses, and evaluation practices within established research. Gap analysis translates prior limitations into concrete design responses such as unified threat constraints and cross-family comparisons. Claims remain within the scope of reviewed evidence, with results written to be comparable and reproducible.

5.4.2 Engineering Activities

Mapping with Complex Engineering Activities

Table 5.3: Mapping with Complex Engineering Activities.

EA1 Range of re-sources	EA2 Level of Interaction	EA3 Innovation	EA4 Consequences for society and environment	EA5 Familiarity
√	√	√		√

5.4.2.1 Justification for Engineering Activities Mapping:

- EA1 Range of resources: Large labeled corpora, high-dimensional features, and hosted graphics processing resources are required. Scientific libraries, training checkpoints, indices, and adversarial arrays form the working asset base. The scholarly record guides method selection and reporting.
- EA2 Level of interaction: Regular supervisor reviews align scope, risks, and reporting standards. Peer feedback and test-slice dry runs refine attack settings and detector thresholds. Results are communicated through manifests, tables, and traceable figures.

- EA3 Innovation: An end-to-end, attack-aware pipeline integrates adversarial training, squeezing-based detection, and simple ensembling. Unified projection rules across models enable fair comparison rarely seen in single-model studies. Artifact discipline turns a research prototype into a reproducible baseline.
- EA5 Familiarity: Some steps are standard (data prep, baseline training), but adaptation to adversarial settings is non-routine. Custom attack harnesses, domain-bounded projections, and detector calibration demand original engineering. Operational framing requires explicit trade-offs rather than benchmark-only goals.

5.5 Summary

This chapter tied the pipeline to established software, hardware, and communication standards and explained how those standards supported safety, reproducibility, and security. It considered the effects on people and organizations, set ethical boundaries, and described a sustainability plan for compute and artefacts. It recorded the management approach and budget. Finally, it demonstrated that adversarial malware detection meets recognized criteria for complex engineering problems and activities and showed how our methods addressed that complexity [28], [29].

Chapter 6

Conclusion

This chapter summarizes what we built and measured, emphasizing the main findings and what they imply for dependable screening. It states the limitations with focus on computational and resource constraints and on methodological scope. It then outlines concrete future work that strengthens training, broadens evaluation, and extends the system toward more resilient practice.

6.1 Summary

We set out to design, implement, and evaluate a static malware detection system that remained dependable in the presence of an adversary. We represented Windows Portable Executable files as fixed-length feature vectors and trained three complementary model families. Light Gradient Boosting Machine produced the strongest clean baseline at 97.63% accuracy and 99.62% receiver operating characteristic area under the curve on the full test set, followed by the one-dimensional convolutional neural network at 96.37% accuracy and 99.24% area under the curve, and the feed forward neural network at 95.68% accuracy and 98.48% area under the curve. We then exercised these models with bounded feature-space attacks that respected plausible constraints for executable validity. Iterative projected gradient descent was most damaging for the neural networks, reducing the one-dimensional convolutional neural network to 0.21% accuracy at $\varepsilon = 0.50$ and 0.23% at $\varepsilon = 1.00$ on the aligned one hundred thousand sample slice, while the feed forward neural network reached 14.42% at $\varepsilon = 0.05$ and 28.43% at $\varepsilon = 0.10$ [5]. A decision-guided routine moved Light Gradient Boosting Machine to 43.37% accuracy under a label-only setting with a strict query budget, which is consistent with decision-based black-box attacks against non-differentiable learners.

Defensive components yielded mixed results. Adversarial training for the feed forward neural network reduced clean performance on the aligned slice to 65.40% and did not retain accuracy under the strongest iterative settings we tested, which matches the documented robustness–accuracy trade-off for adversarial training [6]. Detection based on feature squeezing offered a practical signal at a fixed 1% false-positive rate for several attack sets, with the ensemble detector reaching 44.00% true-positive rate on the hardest convolutional attacks while remaining far less sensitive to the strongest feed forward projected gradient descent [16]. A simple probability-averaging ensemble moderated model-specific failure and stabilized clean predictions, but it did not overturn strong iterative attack success, in line with prior observations that ensembling improves stability rather than providing a certifying defense [10]. All experiments were seed-controlled and artifact-backed so that figures and tables can be traced to saved scalers, checkpoints, indices, adversarial arrays, and metrics.

6.2 Limitation

The primary limitation was computational and resource related. The environment provided a single mid-range graphics processing unit with twelve gigabytes of device memory. This constrained the breadth of hyperparameter sweeps, the number of training restarts, and the scope of multi-step adversarial training. We limited iterative attacks to 10 steps and capped the label-only procedure at 300 queries to keep runtimes tractable. To balance cost and coverage, robustness analyses used a fixed one hundred thousand sample slice rather than the full two hundred thousand sample test set. Memory limits led us to rely on streaming through memory-mapped arrays, which is reliable but slows repeated passes during large grid searches. These choices are common in robustness studies conducted under bounded compute [5].

Methodologically, several decisions reflect these constraints. The defended model used fast gradient sign method-based adversarial training rather than stronger projected gradient descent-based training or a trade-off regularization scheme aligned to the evaluation budgets [6]. The training budget did not include a multi-epsilon curriculum, and the perturbation budgets were not fully unified across all model families. Detectors were calibrated at a single 1% false-positive operating point, which is informative but does not show the full trade-off curve that operators may require [10]. The study remained within a static analysis setting. We did not integrate dynamic behavior features, measure transfer between dynamic and static spaces, or attempt certified robustness methods. Finally, many results were reported under a single seed to preserve runtime, which reduces the strength of statistical claims about variance.

6.3 Future Work

Future work should address both the computational limits and the methodological gaps identified here. A stronger training protocol for the neural models is the first priority. Projected gradient descent-based adversarial training or a trade-off regularization approach with a multi-epsilon curriculum, longer inner loops, and random starts should be aligned to the exact evaluation budgets. Early stopping should track a robustness-aware metric together with clean accuracy so that the operating point is chosen explicitly [5], [6]. Detector design can be extended by combining complementary transformations and by reporting full receiver operating characteristic curves at multiple operating points, for example 1%, 2%, and 5% false-positive rates, so that operational choices are visible [10]. Light Gradient Boosting Machine would benefit from structure-aware defenses or a gating strategy that routes detector-flagged samples to a conservative decision near split thresholds, rather than relying on the raw model alone.

Evaluation should be broadened. Multiple seeds, limited cross-validation, and unified perturbation budgets across model families would make comparisons tighter and more robust. Adaptive and transfer attacks can test whether detection signals survive when an attacker optimizes against them. Where resources allow, the attack step count and the black-box query budget should be increased to probe the steep part of the failure curve.

Extending the pipeline to include dynamic signals or hybrid static and dynamic features is a direct path to a more robust detector. Certified robustness methods, even at small budgets, would add a principled bound that complements empirical attacks. These steps build on what we learned. Clean baselines were strong and the pipeline was reproducible. The robustness gaps are now explicit, along with the resource levers that control them. Addressing those gaps with stronger training, broader evaluation, and modest hardware upgrades would move the system from a diagnostic baseline toward a dependable first line of defense.

References

[1] M. Salman, B. Z. H. Zhao, H. Asghar, and others, “On the Robustness of Malware Detectors to Adversarial Samples,” 2024.

[2] O. Suciu, S. E. Coull, and J. Johns, “Exploring Adversarial Examples in Malware Detection,” 2019.

[3] H. S. Anderson and P. Roth, “EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models,” 2018.

[4] M. Abedin and T. Mehrub, “Evaluating Ensemble and Deep Learning Models for Static Malware Detection with Dimensionality Reduction Using the EMBER Dataset,” 2025.

[5] W. Villegas-Ch, A. Jaramillo-Alcázar, and S. Luján-Mora, “Evaluating the Robustness of Deep Learning Models Against Adversarial Attacks: An Analysis with FGSM, PGD and CW,” 2024.

[6] M. Zhao, L. Zhang, J. Ye, and others, “Adversarial Training: A Survey,” 2020.

[7] I. Rosenberg, A. Shabtai, Y. Elovici, and others, “Adversarial Machine Learning Attacks and Defense Methods in the Cyber Security Domain,” 2021.

[8] H. Zhang, J. Liu, and J. Dong, “CARE: Ensemble Adversarial Robustness Evaluation Against Adaptive Attackers for Security Applications,” 2024.

[9] K. Ren, T. Zheng, Z. Qin, and others, “Adversarial Attacks and Defenses in Deep Learning,” 2020.

[10] W. Xu, D. Evans, and Y. Qi, “Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks,” 2018.

[11] M. Kozák, M. Jureček, M. Stamp, and others, “Creating Valid Adversarial Examples of Malware,” 2023.

[12] D. Gibert, G. Zizzo, Q. Le, and others, “A Robust Defense Against Adversarial Attacks on Deep Learning-Based Malware Detectors via (De)Randomized Smoothing,” 2024.

[13] P. M. D. Sai, L. Amasala, T. S. Bhimavarapu, and others, “A Malware Classification Survey on Adversarial Attacks and Defenses,” 2023.

[14] L. Li, “Comprehensive Survey on Adversarial Examples in Cybersecurity: Impacts, Challenges, and Mitigation Strategies,” 2024.

[15] Y. Wang, J. Liu, X. Chang, and others, “AB-FGSM: AdaBelief Optimizer and FGSM-Based Approach to Generate Adversarial Examples,” 2022.

[16] H. Xu, Y. Ma, H. Liu, and others, “Adversarial Attacks and Defenses in Images, Graphs and Text: A Review,” 2019.

[17] W. Brendel, J. Rauber, and M. Bethge, "Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models," International Conference on Learning Representations, 2018.

[18] ISO/IEC/IEEE 12207:2017, "Systems and Software Engineering — Software Life Cycle Processes," 2017.

[19] IEEE Std 1012-2016, "IEEE Standard for System, Software, and Hardware Verification and Validation," 2016.

[20] NIST, "Security and Privacy Controls for Information Systems and Organizations," Special Publication 800-53, Rev. 5, 2020.

[21] ISO/IEC 27001:2022, "Information Security, Cybersecurity and Privacy Protection — Information Security Management Systems — Requirements," 2022.

[22] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, Aug. 2018.

[23] Association for Computing Machinery, "Artifact Review and Badging — Version 1.1," 2020.

[24] G. van Rossum, B. Warsaw, and N. Coghlan, "PEP 8: Style Guide for Python Code," Python Software Foundation, 2001.

[25] T. Preston-Werner, "Semantic Versioning 2.0.0," 2013.

[26] International Engineering Alliance, "Graduate Attributes and Professional Competencies," 2021.

[27] D. Dittrich and E. Kenneally, "The Menlo Report: Ethical Principles Guiding Information and Communication Technology Research," U.S. Dept. of Homeland Security, 2012.

[28] Association for Computing Machinery, "ACM Code of Ethics and Professional Conduct," 2018.

[29] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green AI," Communications of the ACM, vol. 63, no. 12, pp. 54–63, 2020.

17.09.2025

18% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Match Groups

- **109 Not Cited or Quoted 16%**
  Matches with neither in-text citation nor quotation marks
- **4 Missing Quotations 0%**
  Matches that are still very similar to source material
- **18 Missing Citation 2%**
  Matches that have quotation marks, but no in-text citation
- **0 Cited and Quoted 0%**
  Matches with in-text citation present, but no quotation marks

Top Sources

- 14% Internet sources
- 8% Publications
- 17% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

Match Groups

- 109 Not Cited or Quoted 16%
  Matches with neither in-text citation nor quotation marks
- 4 Missing Quotations 0%
  Matches that are still very similar to source material
- 18 Missing Citation 2%
  Matches that have quotation marks, but no in-text citation
- 0 Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

- 14% Internet sources
- 8% Publications
- 17% Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Submitted works	
Daffodil International University on 2024-12-28		4%
2	Internet	
dspace.daffodilvarsity.edu.bd:8080		3%
3	Internet	
arxiv.org		<1%
4	Submitted works	
Daffodil International University on 2024-12-25		<1%
5	Submitted works	
Heriot-Watt University on 2020-04-22		<1%
6	Internet	
nicholas.carlini.com		<1%
7	Submitted works	
Heriot-Watt University on 2024-03-31		<1%
8	Internet	
www.scirp.org		<1%
9	Submitted works	
University of Finance – Marketing on 2024-08-29		<1%
10	Publication	
Liangheng Zhang, Congmei Jiang, Aiping Pang. "Black-box attacks and defense fo...		<1%

*% detected as AI

AI detection includes the possibility of false positives. Although some text in this submission is likely AI generated, scores below the 20% threshold are not surfaced because they have a higher likelihood of false positives.

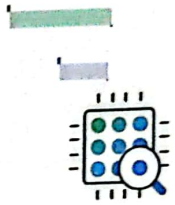
Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Disclaimer
Our AI writing assessment is designed to help educators identify text that might be prepared by a generative AI tool. Our AI writing assessment may not always be accurate (it may misidentify writing that is likely AI generated as AI generated and AI paraphrased or likely AI generated and AI paraphrased writing as only AI generated) so it should not be used as the sole basis for adverse actions against a student. It takes further scrutiny and human judgment in conjunction with an organization's application of its specific academic policies to determine whether any academic misconduct has occurred.

Frequently Asked Questions

How should I interpret Turnitin's AI writing percentage and false positives?
The percentage shown in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was either likely AI-generated text from a large-language model or likely AI-generated text that was likely revised using an AI paraphrase tool or word spinner.



False positives (incorrectly flagging human-written text as AI-generated) are a possibility in AI models.

AI detection scores under 20%, which we do not surface in new reports, have a higher likelihood of false positives. To reduce the likelihood of misinterpretation, no score or highlights are attributed and are indicated with an asterisk in the report (*%).

The AI writing percentage should not be the sole basis to determine whether misconduct has occurred. The reviewer/instructor should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in accordance with their school's policies.

What does 'qualifying text' mean?
Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be likely AI-generated will be highlighted in cyan in the submission, and likely AI-generated and then likely AI-paraphrased will be highlighted purple.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown.