Secure AI Systems
CS5913 - Security in Intelligent Systems

Name: Afzaal Ahmad
Roll No: CS24MTECH02002

Department of Computer Science
Indian Institute Of Technology Hyderabad

November 22, 2025

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Abstract

This report presents a comprehensive Red Team / Blue Team analysis of a Convolutional Neural Network (CNN) trained on the MNIST handwritten digit dataset. Attacks include data poisoning (backdoor injection) and FGSM adversarial perturbations. Defense is implemented using adversarial training. Additionally, we conduct STRIDE threat modeling and Static Application Security Testing (SAST) using Bandit. Through baseline evaluation, attack demonstrations, and defense strategies, we analyze the robustness and security posture of the MNIST classifier.

Contents

1 Introduction

Machine learning has become an essential component of modern systems in authentication, medical diagnosis, autonomous navigation, digital forensics, and countless safety-critical applications. However, conventional machine learning pipelines are not inherently secure. Attackers can compromise the system at multiple stages: by altering training data, injecting adversarial noise, or manipulating model inputs.

This assignment investigates the security of a Convolutional Neural Network (CNN) trained on the MNIST dataset through adversarial experiments. We take a dual Red Team (attack) and Blue Team (defense) perspective and evaluate:

- Baseline performance on clean MNIST data.
- Data poisoning via a backdoor trigger.
- Adversarial examples generated using FGSM.
- Defense through adversarial training.
- STRIDE threat modeling for holistic system-level security.
- Static code security testing with Bandit (SAST).

Our experiments demonstrate that models achieving near-perfect accuracy can still be highly vulnerable to targeted manipulation. By analyzing these vulnerabilities and applying defensive strategies, we gain insights into secure AI system design.

2 Dataset and Preprocessing

2.1 MNIST IDX Format

The MNIST dataset consists of:

- `train-images.idx3-ubyte` (60,000 images)
- `train-labels.idx1-ubyte` (60,000 labels)
- `t10k-images.idx3-ubyte` (10,000 images)
- `t10k-labels.idx1-ubyte` (10,000 labels)

We load these binary IDX files manually to comply with assignment instructions and retain full control over data handling. This prevents hidden preprocessing steps and allows insertion of poisoning triggers at a low level.

2.2 Normalization and Conversion

After loading raw bytes:

- Pixel values are normalized to $[0, 1]$.
- Images are reshaped into $(1, 28, 28)$ tensors.
- Labels are integer encoded.

2.3 Data Integrity Checks

Instead of unsafe `assert` statements, explicit integrity checks raise descriptive errors if:

- image and label counts mismatch,
- dimensions are incorrect,
- label values fall outside the expected range.

This mitigates accidental corruption and strengthens pipeline security.

3 CNN Model Architecture

The MNIST classifier uses the following CNN:

- Conv2d: $1 \to 32$, 3×3 kernel, padding=1
- Conv2d: $32 \to 64$, 3×3 kernel, padding=1
- MaxPool2d: 2×2
- Fully Connected: $64 \cdot 14 \cdot 14 \to 128$
- Fully Connected: $128 \to 10$
- Activation: ReLU

Design choices:

- Small kernels capture fine digit strokes.
- Max pooling downsamples and increases invariance.
- ReLU reduces vanishing gradient issues.

This model balances accuracy (99%) and vulnerability (ideal for adversarial experiments).

4 Experimental Modes

We structured experiments into four modes to isolate effects:

4.1 Baseline

Trains the CNN on clean MNIST and serves as the reference model for all comparisons.

4.2 Poisoned Mode

Implements a backdoor attack where:

- A white 4×4 square trigger is added to 100 images of digit "7".
- These images are relabeled as digit "1".

This simulates a realistic dataset poisoning scenario.

4.3 Adversarial (FGSM) Mode

Generates adversarial samples using FGSM:

$$x_{\text{adv}} = x + \epsilon \cdot \text{sign}(\nabla_x L(x, y))$$

This models evasion attacks during inference.

4.4 Adversarial Training Mode

Hardens the model by:

- generating FGSM adversarial samples in each batch,
- training on both clean and adversarial data.

5 Baseline Performance

5.1 Results

- Test Loss: 0.0400
- Test Accuracy: 98.91%
- Inference Time: $\approx$ 5.94 ms/image

Table 1: Baseline Model Performance

Metric	Value
Accuracy	98.91%
Loss	0.0400
Inference Time	5.94 ms/image

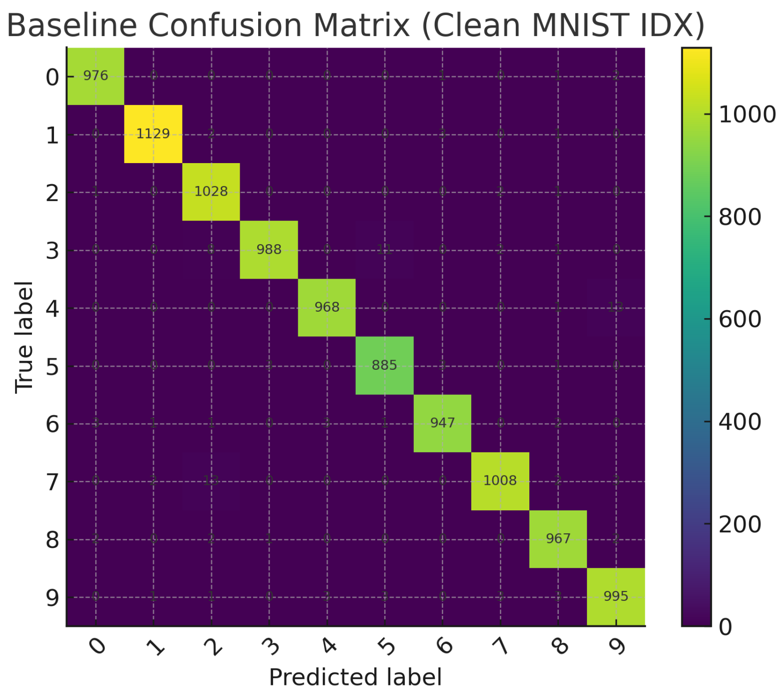

Figure 1: Baseline confusion matrix.

The model performs uniformly well across all classes, with few misclassifications.

6 Data Poisoning Attack

Data poisoning modifies the training dataset to implant hidden malicious behavior.

6.1 Backdoor Trigger Method

- Add a small white square to 7s.
- Change their label to 1.

6.2 Findings

- Clean accuracy remains high (98.69%) — attack remains stealthy.
- Triggered 7s are misclassified as 1 with high confidence.

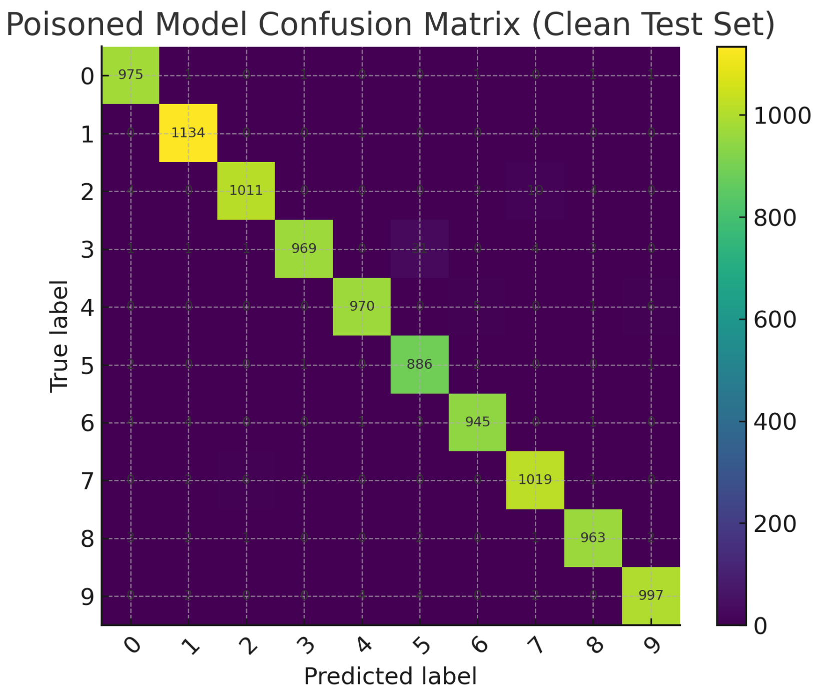

Figure 2: Poisoned test confusion matrix.

This demonstrates a powerful backdoor attack with minimal poisoned samples.

7 FGSM Adversarial Attack

FGSM exploits gradients to perturb the input image in the direction of maximum loss.

7.1 Results

- Clean accuracy: 99.07%
- FGSM accuracy: 17.50%

The drastic accuracy drop highlights vulnerability to even small perturbations.

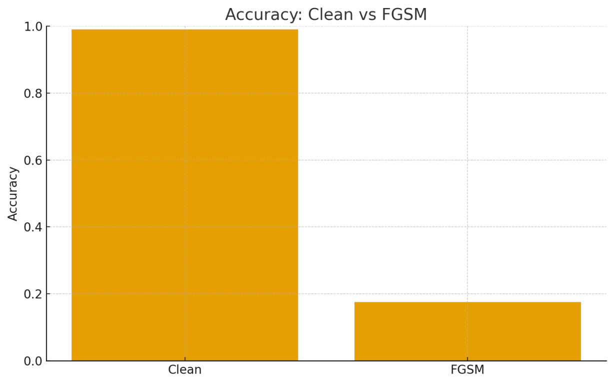

Figure 3: Accuracy of the baseline model on clean vs FGSM adversarial samples.

8 Adversarial Training Defense

Adversarial training is implemented to improve robustness.

8.1 Results

Before defense:

- Clean: 98.91%
- FGSM: 17.50%

After defense:

- Clean: 98.96%
- FGSM: 81.56%

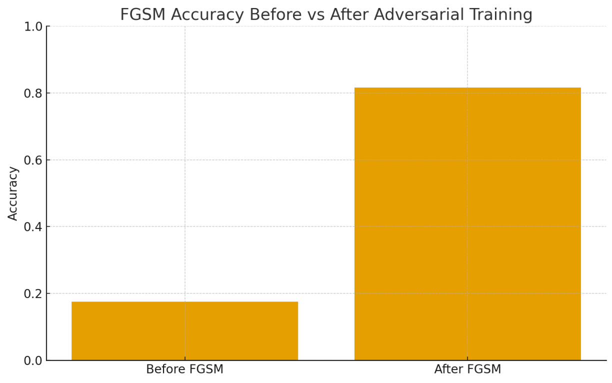

Figure 4: FGSM accuracy before vs after adversarial training.

Adversarial training maintains clean accuracy and dramatically increases robustness.

9 Threat Modeling using STRIDE

We apply STRIDE to understand system-level vulnerabilities and how our implementation aligns with mitigating them.

Spoofing

Potential impersonation of legitimate clients or data sources. Our implementation uses explicit data set paths and formats, preventing unverified data ingestion.

Tampering

Attackers may modify training data. Our poisoning module demonstrates this risk. Integrity checks prevent accidental mismatches.

Repudiation

Attackers may deny malicious actions. Structured training modes (baseline, poisoned, adv_only, adv_training) serve as an audit trail.

Information Disclosure

Model or data exposure can enable stronger attacks. All processing is local; no gradients or weights are transmitted externally.

Denial of Service

Expensive adversarial inputs can slow inference. We measure inference time and evaluate performance under adversarial load.

Elevation of Privilege

Unauthorized retraining can install backdoors. Controlled dataset wrappers isolate poisoning logic and enforce explicit intent.

10 Static Application Security Testing (SAST)

Bandit was used to scan the entire Python codebase:

- 1 low-severity issue detected: unsafe `assert`.
- Fixed by replacing with a proper `ValueError`.
- No high or medium issues after remediation.

SAST confirms secure coding practices across the implementation.

11 Conclusion

This project shows that:

- High accuracy (≈99%) does not imply robustness.
- Minimal poisoning successfully implants a covert backdoor.
- FGSM adversarial examples reduce accuracy drastically.

- Adversarial training significantly improves robustness (17.5% $\rightarrow$ 81.56% FGSM accuracy).
- STRIDE reveals additional systemic vulnerabilities beyond accuracy.
- SAST ensures strong code hygiene and reduces implementation risks.

The Red/Blue Team methodology provides a holistic understanding of adversarial machine learning and secure AI system design.

12 Github Repo

- GitHub repository of CNN implementation for MNIST classification: `https://github.com/ahmadafzaal96/mnist-secure-ai/blob/244ca47ebefdc40164f842180bc4c2b55f7f26db/src/mnist_secure_cnn_idx.py`
- GitHub repository for adversarial dataset generation: (Line Number - 238) `https://github.com/ahmadafzaal96/mnist-secure-ai/blob/244ca47ebefdc40164f842180bc4c2b55f7f26db/src/mnist_secure_cnn_idx.py`