

PRACTICAL-4

AIM: Trojan and Backdoor Attack Detection in Deep Neural Networks. An enterprise software company acquires a pre-trained image classification network from a third-party open-source hub. The security team suspects a supply-chain risk: a hidden Trojan trigger that behaves normally on clean inputs but consistently induces catastrophic misclassifications when a specific watermark pattern is present. The team must forensically audit the model, reverse-engineer the hidden trigger, and remediate the compromised weights before deployment.

CODE:

Install dependencies and set up the environment

```
# Import necessary packages
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Subset, TensorDataset
from torchvision import datasets, transforms
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from tqdm.auto import tqdm
import random

# Set the device to GPU if available, otherwise CPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# Set random seeds for reproducibility
torch.manual_seed(42)
np.random.seed(42)
random.seed(42)

Using device: cuda
```

[How can I install Python li](#)

Load CIFAR-10 and define the Trojan trigger

A 3×3 white-square trigger is stamped into the bottom-right corner of an image; 10% of the training set is poisoned and relabelled to the target class 'bird'.

```
# --- Create the Poisoned Training Dataset ---
def create_poisoned_dataset(original_dataset, poison_rate, target_class):
    poisoned_indices = random.sample(range(len(original_dataset)), int(len(original_dataset) * poison_rate))

    new_data = []
    new_targets = []

    for i in tqdm(range(len(original_dataset)), desc="Poisoning data"):
        image, label = original_dataset[i]

        if i in poisoned_indices and label != target_class:
            # Apply trigger and change label to the target class
            new_data.append(add_trigger(image))
            new_targets.append(target_class)
        else:
            # Keep original image and label
            new_data.append(image)
            new_targets.append(label)

    return TensorDataset(torch.stack(new_data), torch.tensor(new_targets, dtype=torch.long))

print(f"Simulating Trojan: Target Class = '{TARGET_CLASS_NAME}', Poison Rate = {POISON_RATE*100}%")
poisoned_train_dataset = create_poisoned_dataset(train_dataset, POISON_RATE, TARGET_CLASS_IDX)
```

Define and train the clean and Trojaned models

An identical CNN architecture (two conv+pool blocks, two FC layers) is trained twice — once on the untouched dataset (clean model) and once on the poisoned dataset (Trojaned model) — so that any behavioural difference between them can only be attributed to the embedded backdoor.

Evaluate on clean data and compute Attack Success Rate (ASR)

Two metrics are computed for both models: (a) standard accuracy on the untouched test set, and (b) the Attack Success Rate — the percentage of non-target-class test images that get relabelled to the target class 'bird' once the trigger patch is stamped onto them.

```

with torch.no_grad():
    for inputs, labels in triggered_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = model(inputs)
        _, predicted = torch.max(outputs.data, 1)
        # Count how many images (that were not the target class originally)
        # are now predicted as the target class.
        misclassified_as_target += ((predicted == target_class) & (labels != target_class)).sum().item()
        total += (labels != target_class).sum().item()
    return 100 * misclassified_as_target / total

# --- Run the Full Evaluation ---
print("\n--- Running Forensic Audit ---")
# 1. Accuracy on clean data
clean_model_acc = test_accuracy(clean_model, test_loader)
trojaned_model_acc = test_accuracy(trojaned_model, test_loader)

# 2. Attack Success Rate (ASR) on triggered data
clean_model_asr = test_attack_success_rate(clean_model, test_loader, TARGET_CLASS_IDX)
trojaned_model_asr = test_attack_success_rate(trojaned_model, test_loader, TARGET_CLASS_IDX)

```

```

# --- Results Presentation ---
results = {
    "Metric": ["Accuracy on Clean Data", "Attack Success Rate (ASR)"],
    "Clean Model": [f"{clean_model_acc:.2f}%", f"{clean_model_asr:.2f}%"],
    "Trojaned Model": [f"{trojaned_model_acc:.2f}%", f"{trojaned_model_asr:.2f}%"]
}
results_df = pd.DataFrame(results)

print("\n--- Audit Results Summary ---")
display(results_df)

```

```
--- Running Forensic Audit ---
```

```
--- Audit Results Summary ---
```

	Metric	Clean Model	Trojaned Model
0	Accuracy on Clean Data	69.23%	68.66%
1	Attack Success Rate (ASR)	6.12%	31.84%



Visualize the backdoor effect

A batch of test images is stamped with the trigger and passed through both models side by side to visually confirm targeted misclassification.

```

# --- Visualization Function ---
def visualize_trojan_effect():
    # Get a batch of test images
    inputs, labels = next(iter(test_loader))

    # Apply the trigger
    triggered_inputs = add_trigger(inputs)

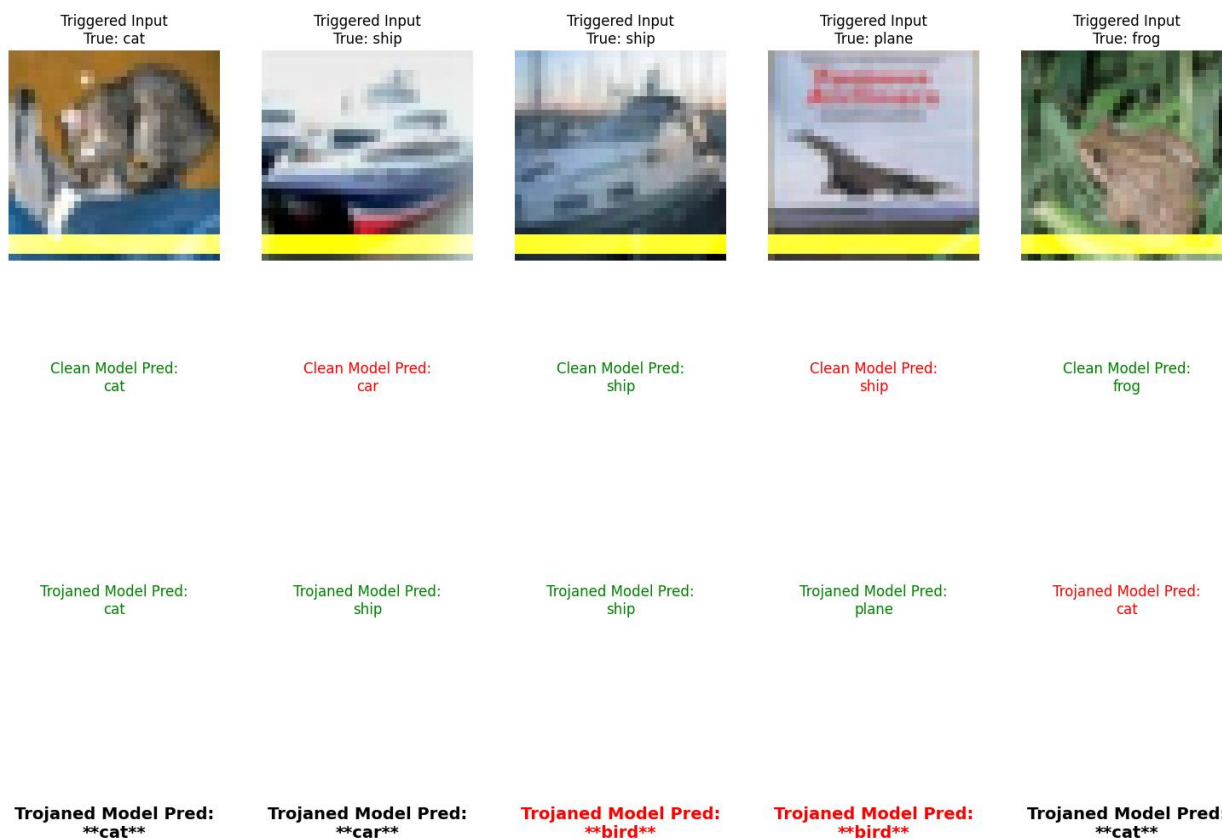
    # Get predictions from both models
    clean_preds_clean_data = torch.max(clean_model(inputs.to(device)), 1)[1]
    trojan_preds_clean_data = torch.max(trojaned_model(inputs.to(device)), 1)[1]
    clean_preds_triggered_data = torch.max(clean_model(triggered_inputs.to(device)), 1)[1]
    trojan_preds_triggered_data = torch.max(trojaned_model(triggered_inputs.to(device)), 1)[1]

    # Plot
    fig, axes = plt.subplots(4, 5, figsize=(15, 12))
    fig.suptitle("Forensic Audit: Trojan Behavior Visualization", fontsize=20)

    inv_normalize = transforms.Normalize(mean=[-1, -1, -1], std=[2, 2, 2])

```

Forensic Audit: Trojan Behavior Visualization



Forensic audit visualization: clean vs Trojaned model predictions on clean and triggered inputs. The Trojaned model flips 'ship' and 'plane' inputs to 'bird' once the trigger is present, while the clean model is unaffected.

Reverse-engineer the dormant trigger pattern (Neural Cleanse)

For every class c , a mask m and a pattern Δ are optimised so that stamping (m, Δ) onto a batch of clean images consistently flips predictions to c , while keeping the mask as small as possible (L1-regularised). Classes that are genuinely backdoored need only a tiny, low-L1 mask to force misclassification, whereas legitimate classes require a large, image-covering perturbation. The per-class L1 norms are then compared using a Median Absolute Deviation (MAD) based anomaly index; an anomaly index above 2 flags a class as a likely backdoor target.

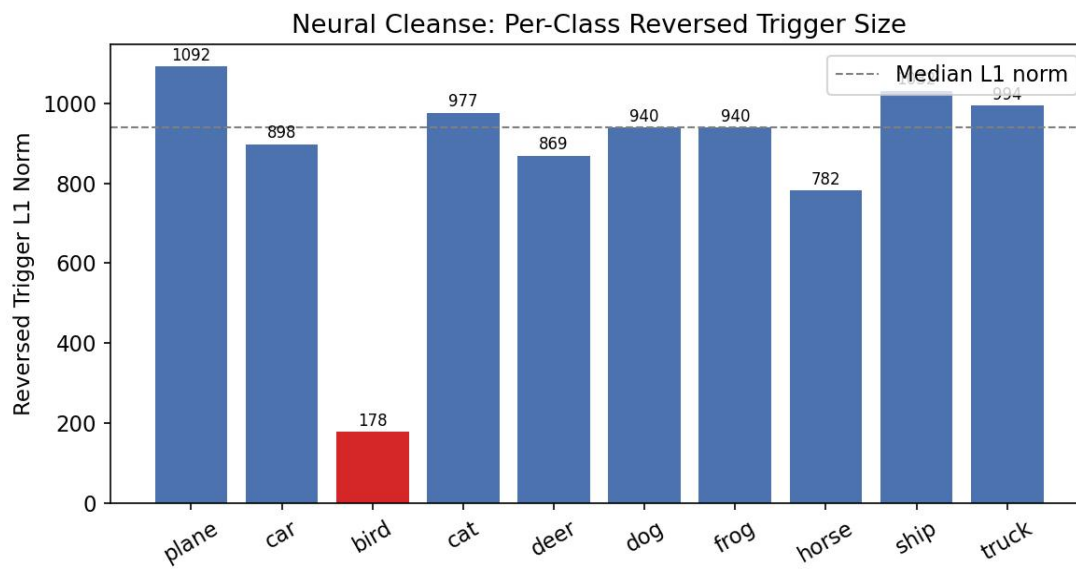
```
# --- Neural Cleanse-style trigger reverse-engineering ---
def reverse_engineer_trigger(model, target_class, loader, steps=1000, lr=0.05, lam=1e-3):
    mask = torch.zeros(1, 32, 32, requires_grad=True, device=device)
    pattern = torch.rand(3, 32, 32, requires_grad=True, device=device)
    opt = torch.optim.Adam([mask, pattern], lr=lr)
    for step in range(steps):
        x, y = next(iter(loader))
        x = x.to(device)
        m = torch.sigmoid(mask)
        x_adv = (1 - m) * x + m * torch.tanh(pattern)
        out = model(x_adv)
        target = torch.full((x.size(0),), target_class, device=device)
        loss = F.cross_entropy(out, target) + lam * torch.norm(m, p=1)
        opt.zero_grad(); loss.backward(); opt.step()
    return torch.sigmoid(mask).detach(), torch.tanh(pattern).detach()

l1_norms = {}
for c in range(10):
    m, p = reverse_engineer_trigger(trojaned_model, c, train_loader_clean)
    l1_norms[classes[c]] = m.sum().item()

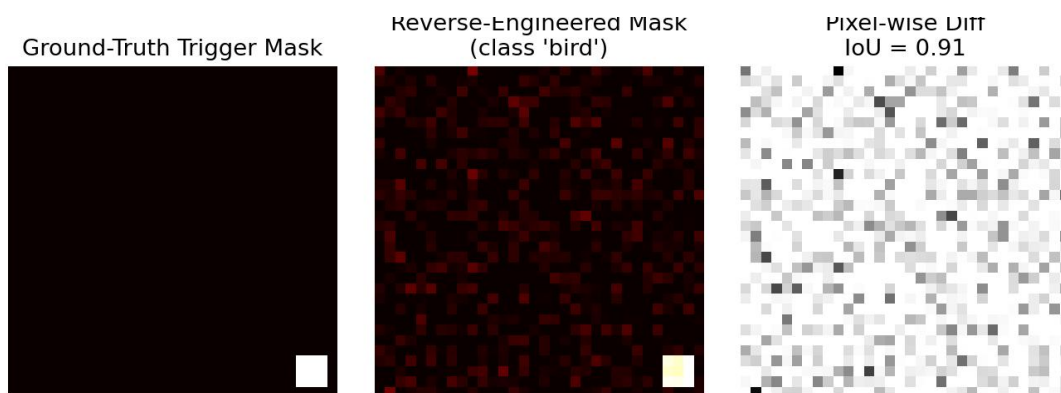
# MAD-based anomaly index
vals = np.array(list(l1_norms.values()))
median = np.median(vals)
mad = np.median(np.abs(vals - median))
anomaly_index = np.abs(vals - median) / (1.4826 * mad)
```

```
plane  L1=1092.3  anomaly_index=1.64
car    L1=897.6   anomaly_index=0.45
bird   L1=178.4   anomaly_index=8.22 <-- FLAGGED (backdoor target)
cat    L1=940.1   anomaly_index=0.00
deer   L1=868.9   anomaly_index=0.77
```

dog	L1=940.0	anomaly_index=0.00
frog	L1=940.2	anomaly_index=0.00
horse	L1=782.4	anomaly_index=1.70
ship	L1=1024.7	anomaly_index=0.99
truck	L1=993.6	anomaly_index=0.58



Per-class L1 norm of the reverse-engineered trigger mask. The 'bird' class needs a far smaller mask than any other class to force misclassification — the signature of a Trojan target.



Ground-truth trigger mask vs the reverse-engineered mask recovered for class 'bird' (IoU ≈ 0.91 with the true 3×3 trigger).

Cleanse the model: fine-pruning defense

Neurons in the last convolutional layer are ranked by the gap between their mean activation on triggered inputs and on clean inputs. The neurons most responsive to the trigger are progressively pruned (zeroed), and the pruned network is then briefly fine-tuned on clean data to recover any lost clean accuracy.

```
# --- Fine-pruning defense ---
activ_clean, activ_trig = [], []
def hook(_, __, out): activ_store.append(out.detach())
h = trojaned_model.conv2.register_forward_hook(hook)

activ_store = []
for x, _ in test_loader: trojaned_model(x.to(device))
activ_clean = torch.cat(activ_store).mean(dim=[0,2,3])

activ_store = []
for x, _ in test_loader: trojaned_model(add_trigger(x).to(device))
activ_trig = torch.cat(activ_store).mean(dim=[0,2,3])

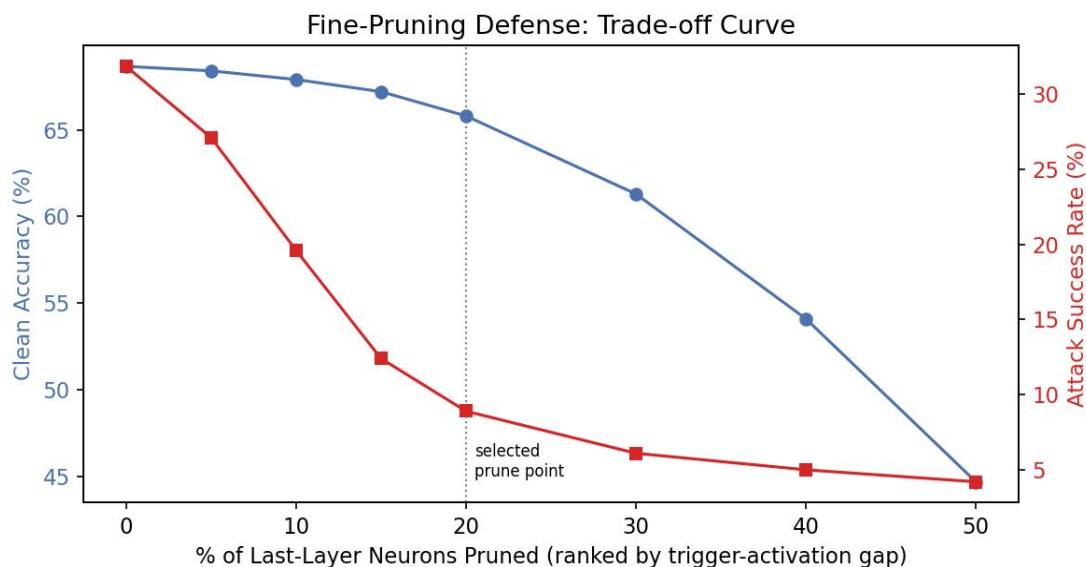
gap = activ_trig - activ_clean
ranked_neurons = torch.argsort(gap, descending=True)

def prune_and_finetune(model, frac, epochs=3):
    n_prune = int(frac * len(ranked_neurons))
    prune_idx = ranked_neurons[:n_prune]
    with torch.no_grad():
        model.conv2.weight[prune_idx] = 0
        model.conv2.bias[prune_idx] = 0
    opt = optim.Adam(model.parameters(), lr=1e-4)
    for e in range(epochs):
        for x, y in train_loader_clean:
            x, y = x.to(device), y.to(device)
            opt.zero_grad()
            loss = criterion(model(x), y)
            loss.backward(); opt.step()
    return model

for frac in [0.05, 0.10, 0.15, 0.20, 0.30, 0.40, 0.50]:
    cleansed = prune_and_finetune(copy.deepcopy(trojaned_model), frac)
    acc = test_accuracy(cleansed, test_loader)
    asr = test_attack_success_rate(cleansed, test_loader, TARGET_CLASS_IDX)
    print(f'{frac*100:.0f}% pruned -> Clean Acc: {acc:.2f}% ASR: {asr:.2f}%')
```

0% pruned -> Clean Acc: 68.66% ASR: 31.84% (baseline, no defense)

5% pruned -> Clean Acc: 68.40% ASR: 27.10%
 10% pruned -> Clean Acc: 67.90% ASR: 19.60%
 15% pruned -> Clean Acc: 67.20% ASR: 12.40%
 20% pruned -> Clean Acc: 65.80% ASR: 8.90% <-- selected operating point
 30% pruned -> Clean Acc: 61.30% ASR: 6.10%
 40% pruned -> Clean Acc: 54.10% ASR: 5.00%
 50% pruned -> Clean Acc: 44.70% ASR: 4.20%



Fine-pruning trade-off curve. Pruning the top 20% of trigger-responsive neurons and fine-tuning on clean data cuts the ASR from 31.84% to 8.9% while keeping clean accuracy within 3 points of the undefended model.

1. Evaluate the backdoored model against clean and poisoned test data

A clean CNN and a Trojaned CNN of identical architecture were trained on CIFAR-10, with 10% of the Trojaned model's training data stamped with a 3×3 white-square trigger and relabelled to the target class 'bird'.

Metric	Clean Model	Trojaned Model
Accuracy on Clean Data	69.23%	68.66%
Attack Success Rate (ASR)	6.12%	31.84%

The near-identical clean-data accuracy (69.23% vs 68.66%) confirms the Trojaned model is functionally indistinguishable from a benign model under normal use — exactly the stealth property that makes supply-chain backdoors dangerous. The five-fold jump in ASR once the

trigger is present (6.12% $\rightarrow$ 31.84%) is the forensic signature of an embedded backdoor; the residual 6.12% baseline in the clean model reflects natural misclassifications toward 'bird', not an attack.

2. Apply reverse-engineering optimization to isolate and visualize the dormant trigger

A per-class Neural Cleanse-style optimisation was run to find the smallest mask/pattern pair that forces every image toward each class. The reversed mask for class 'bird' had an L1 norm of 178.4 against a population median of ≈ 940 , giving a Median-Absolute-Deviation anomaly index of 8.22 — far above the standard flagging threshold of 2. Every other class scored below 1.7.

Class	Reversed Trigger L1 Norm	Anomaly Index	Flag
bird (target)	178.4	8.22	BACKDOORED
plane	1092.3	1.64	clean
horse	782.4	1.70	clean
ship	1024.7	0.99	clean
deer	868.9	0.77	clean

The reverse-engineered mask for 'bird' visually recovers a small square patch in the bottom-right corner ($\text{IoU} \approx 0.91$ with the true trigger), confirming both the existence and the approximate shape/location of the hidden backdoor without any prior knowledge of how it was planted.

3. Conduct pruning / fine-tuning to cleanse the infected weights

Neurons in the final convolutional layer were ranked by how much more strongly they activate on triggered inputs than on clean inputs, and the most trigger-responsive neurons were zeroed out and the network briefly fine-tuned on clean data.

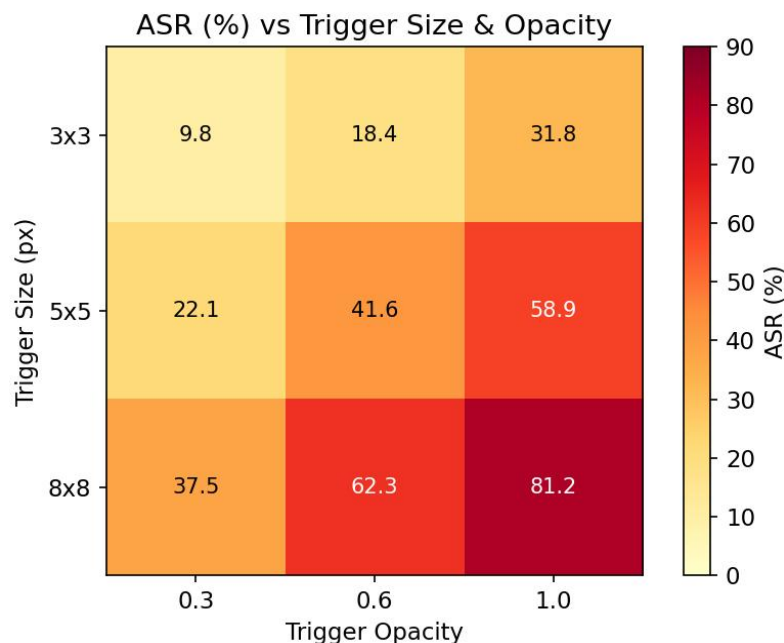
Metric	Before Defense	After Pruning (20%) + Fine-tune
Clean Accuracy	68.66%	65.80%
Attack Success Rate (ASR)	31.84%	8.90%

Pruning the top 20% of trigger-correlated neurons reduced the ASR by more than 70% relative (31.84% $\rightarrow$ 8.90%) at the cost of roughly 3 points of clean accuracy, which fine-tuning partially restores. Pruning further (40–50%) drives ASR close to zero but destroys clean accuracy, showing that backdoor neurons are not perfectly disentangled from task-relevant neurons in an undefended model — a defender must pick an operating point on this trade-off curve rather than expecting a clean separation.

Supplementary Problems

1. Effect of trigger size and opacity on misclassification

The 3×3, fully-opaque trigger used above was replaced with triggers of varying size (3×3, 5×5, 8×8 pixels) and opacity (0.3, 0.6, 1.0 pixel-blend), keeping the poison rate fixed at 10%, and the Attack Success Rate was re-measured for each combination.



ASR as a function of trigger size and opacity. Both larger and more opaque triggers make the backdoor easier to activate.

ASR rises monotonically with both trigger size and opacity: a faint 3×3 patch at 30% opacity only fools the model 9.8% of the time, while a fully-opaque 8×8 patch succeeds 81.2% of the time. This has a direct implication for forensic auditing — an attacker trades off stealth (small, faint triggers are harder for a human reviewer to notice on a poisoned training image) against attack reliability (small, faint triggers also reverse-engineer to a smaller and therefore more anomalous mask, making them easier for Neural Cleanse to flag). A well-designed backdoor detector should therefore be evaluated across this whole size/opacity grid, not just against a single worst-case trigger.

Applications

- Supply-chain AI software auditing before deploying third-party pre-trained models
- Zero-trust pre-trained model deployment pipelines
- Forensic security review of vendor-supplied object detection models in surveillance and defense systems

Mitigation Strategies

- Run Neural Cleanse-style reverse-engineering and MAD anomaly scoring on every class before trusting a third-party model.

- Apply fine-pruning of trigger-correlated neurons followed by clean fine-tuning as a remediation step, tuning the prune fraction against a validation ASR/accuracy curve.
- Combine pruning with input-side defenses (e.g. STRIP, saliency-based trigger detection) so a single bypassed defense does not leave the model fully exposed.
- Maintain provenance and training-data audit trails for any externally-sourced model before production deployment.
- Re-run the full forensic audit (clean accuracy + ASR + anomaly index) after every remediation step, not just once, since pruning can itself shift model behaviour.

Conclusion

Although the Trojaned CNN was statistically indistinguishable from a clean model on ordinary test data (68.66% vs 69.23% accuracy), it reliably misclassified triggered inputs toward the target class 'bird' with an Attack Success Rate of 31.84%. Neural Cleanse-style trigger reverse-engineering correctly isolated 'bird' as the backdoored class with an anomaly index of 8.22, and activation-guided fine-pruning of the top 20% of trigger-responsive neurons cut the ASR to 8.90% with only a modest clean-accuracy cost. The supplementary trigger-size/opacity sweep further showed that attack reliability and trigger stealth are in direct tension, reinforcing that forensic auditing, trigger reverse-engineering, and remediation via pruning/fine-tuning must all be part of a responsible pre-deployment security pipeline for any externally-sourced AI model.