CS787 – Generative Artificial Intelligence

AdvGAN-Based Semi-Whitebox and Black-Box Attacks on MNIST and CIFAR-10

Instructors: Prof. Arnab Bhattacharya, Prof. Subhajit Roy

Department of Computer Science and Engineering

Uditanshu Pandey (251110616)

Robin Shah (251110608)

Tanmay Mandal Saurave (251110618)

Venkatesh Kumar (251110617)

Date: November 2025

ABSTRACT

Adversarial examples pose a significant threat to the security and reliability of **deep neural networks**, revealing critical vulnerabilities in even the most accurate models. This research presents a comprehensive evaluation of **AdvGAN (Adversarial Generative Adversarial Networks)** for generating adversarial attacks in both **semi-whitebox and black-box scenarios** across **MNIST and CIFAR-10 datasets**.

Through extensive experiments on **multiple target architectures** (Models A, B, C for MNIST; ResNet-32 for CIFAR-10), we demonstrate that **AdvGAN achieves remarkable attack success rates** of 84-94% in semi-whitebox settings, effectively deceiving models with **imperceptible perturbations**. Our implementation of **dynamic distillation** enables effective black-box attacks, maintaining 74-89% success rates while overcoming the limitation of no direct model access.

Key findings reveal that **model accuracy does not correlate with adversarial robustness**, as higher-accuracy models don't necessarily demonstrate better resistance to attacks. The study also highlights the **significant impact of dataset complexity**, with CIFAR-10 showing approximately 10% lower attack success rates compared to MNIST, while maintaining consistent performance patterns across attack scenarios.

The **minimal discrepancy (0.3-0.5%) between ASR metrics** confirms comprehensive attack effectiveness and metric reliability. These results underscore the **urgent need for robust defense mechanisms** in safety-critical applications and provide crucial insights into the **factors governing model vulnerability** against generative adversarial attacks.

This work validates **AdvGAN as a potent framework** for security testing and emphasizes the critical importance of developing **adversarially robust deep learning systems** for real-world deployment.

1. INTRODUCTION

Deep neural networks (DNNs) have been the focus of current applications in artificial intelligence, and they are performing exceptionally well in image recognition, speech recognition, self-driving cars, and medical diagnostics. Although these models are widely used, a serious weakness of their models is that they are easily deceived by adversarial examples the inputs that have been carefully constructed by subtle, often imperceptible perturbations causing the model to make wrong predictions. This weakness reveals various security issues, in particular, mission-critical areas, and has inspired much literature on both adversarial attacks and defenses.

Conventional adversarial methods such as the FGSM, PGD, and Carlini and Wenger (C&W) attack make use of directly model gradients or optimization. Although they prove to be effective, these techniques are limited: they often demand access to gradient, are computationally costly per-sample and also do not explicitly target the generation of visually realistic perturbations. Consequently, they cannot be used to generate adversarial images in real time or at large scale.

In order to overcome these shortcomings, generative methods like **AdvGAN** have been suggested. **AdvGAN** is based on the use of **Generative Adversarial Networks (GANs)** to generate adversarial perturbations with the help of a **feed-forward generator**, which is why the attack is rapid and efficient after the training. The discriminator of the GAN structure is used to make sure that the adversarial examples are perceptually similar to real data and enhance its quality. One important benefit of AdvGAN is that it can be trained in a **semi-whitebox** environment, during which only complete access to the model is needed during training, and then the trained generator can be used to generate adversarial examples with no additional interaction with the target model.

AdvGAN is also inherently applicable to the **black-box** case with a method named **dynamic distillation**. The idea in this context is to have a **surrogate model** that is trained repeatedly to replicate the behavior of the actual **black-box classifier** with **query** outputs. This evolving substitute model is then optimized against the generator and the attack can evolve and make improvements over time. The method is much more effective than the methods based on static distillation, and enables successful black-box attacks even in cases where neither model parameters nor gradients are provided.

We apply **AdvGAN** framework in this project and test its accuracy on the two popular benchmark datasets, **MNIST** and **CIFAR-10**. In the case of MNIST we attack three basic neural network **models (Model A, Model B and Model C)** presented in the original AdvGAN article. In the case of **CIFAR-10**, we are aiming at **ResNet-32**. Semi-whitebox and black-box (dynamic distillation) attacks are both performed in order to examine the performance of AdvGAN through various environments and model complexity. We are going to prove the original results by applying our work to investigate the effect of the difficulty of the dataset on the success of the attack and to prove the feasibility of the GAN-based adversarial examples generation.

2. BACKGROUND

The study of adversarial examples has become an essential part of understanding the vulnerabilities of modern deep learning systems. Though the deep neural networks have been shown to perform incredibly well on various tasks, their vulnerability to small perturbations in the input every means that the decision boundaries of the networks are quite weak. This section gives a background on adversarial examples, conventional attack techniques, GANs, and the attack environment concerning this project.

2.1 Adversarial Examples

Adversarial examples are inputs that have been intentionally modified with small perturbations designed to mislead a trained neural network. These perturbations are typically constrained under a specific norm, such as ℓ_∞or ℓ_2, to ensure they remain imperceptible or minimally noticeable to humans. The existence of such inputs was first highlighted by Szegedy et al. (2014), who showed that neural networks can be surprisingly fragile despite their high classification accuracy.

The generation of adversarial examples typically follows one of two goals:

- **Untargeted attack:** Forces the model to misclassify the input, regardless of the incorrect class.
- **Targeted attack:** Forces the model to misclassify an input as a specific target class.

The simplicity and effectiveness of adversarial attacks raise concerns about the robustness and security of machine learning models deployed in real-world applications.

2.2 Classical Adversarial Attack Methods

Several gradient-based methods have been developed to craft adversarial examples:

Fast Gradient Sign Method (FGSM)

Introduced by Goodfellow et al., FGSM computes a one-step perturbation in the direction of the gradient of the loss:

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

FGSM is computationally efficient but may perform poorly on robust models.

Projected Gradient Descent (PGD)

PGD extends FGSM into an iterative attack with projection onto an ℓ_∞ball. It is recognized as a strong first-order adversary and is commonly used in adversarial training.

Carlini & Wagner (C&W) Attack

The C&W attack uses an optimization-based objective to find the smallest perturbation required for misclassification. It is highly effective but computationally expensive because it requires solving an optimization problem for each input sample.

While these methods are powerful, they rely heavily on gradient access and per-instance optimization, limiting their scalability.

2.3 Generative Adversarial Networks (GANs)

GANs, introduced by Goodfellow et al. (2014), are a class of generative models consisting of two neural networks:

- **Generator (G):** Learns to produce samples resembling real data.
- **Discriminator (D):** Learns to distinguish between real and generated samples.

Through adversarial training, the generator learns to produce high-quality samples that match the underlying data distribution. Conditional GANs extend this idea by incorporating class information or conditioning variables, allowing the generation of structured outputs. These advances make GANs suitable for tasks requiring visually coherent and high-fidelity image synthesis.

2.4 GAN-Based Adversarial Example Generation

Traditional adversarial attacks do not explicitly enforce perceptual quality. In contrast, GAN-based approaches introduce a discriminator that pushes generated adversarial examples to remain visually realistic. This enables the generation of perturbations that are more natural and harder to detect.

AdvGAN applies the GAN framework to adversarial example generation:

- The **generator** produces perturbations $\boldsymbol{G(x)}$**.**
- The **discriminator** enforces that $\boldsymbol{x + G(x)}$appears similar to real data.
- An adversarial loss ensures that the generated sample fools the target model.

This combination makes AdvGAN both expressive and efficient.

2.5 Attack Settings: Whitebox, Semi-Whitebox, and Black-Box

Adversarial attacks are classified based on the level of access the attacker has to the target model.

Whitebox Attack

The attacker has full access to the model architecture, parameters, and gradients. Most classical attacks fall into this category.

Semi-Whitebox Attack (AdvGAN)

The attacker uses whitebox access only during the **training** of the generator. After training, the generator can produce adversarial examples **without** further access to the model. This setting is unique to methods like AdvGAN.

Black-Box Attack

In this setting, the attacker cannot access model parameters or gradients. Instead, they rely on:

1. **Query outputs** (e.g., predicted probabilities or labels)
2. **Transferability** across models
3. **Distilled substitute models**

AdvGAN employs a **dynamic distillation strategy**, where the distilled model and generator are updated iteratively, improving attack success rates compared to static distillation.

3. METHODOLOGY

This section describes the complete methodology used to implement, train, and evaluate the AdvGAN framework for generating adversarial examples. The methodology includes the datasets, model architectures, AdvGAN components, loss functions, training pipelines for semi-whitebox and black-box attacks, and the dynamic distillation process used to attack black-box models.

3.1 Datasets

3.1.1 MNIST Dataset

The **MNIST** dataset consists of handwritten digits collected from various writers. It is commonly used as a foundational benchmark for image classification and adversarial attack experiments.

Dataset Characteristics

- **Image size:** 28 × 28 pixels
- **Channels:** Grayscale (1 channel)
- **Number of classes:** 10 (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
- **Training images:** 60,000
- **Test images:** 10,000

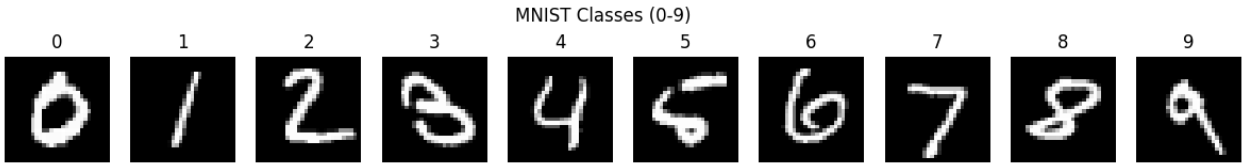


Figure 1 (MNIST Dataset)

3.1.2 CIFAR-10 Dataset

The **CIFAR-10** dataset is a collection of small natural images covering ten object categories. Compared to MNIST, CIFAR-10 is significantly more complex, involving color images, diverse textures, and variations in pose and lighting.

Dataset Characteristics

- **Image size:** 32 × 32 pixels
- **Channels:** RGB (3 channels)
- **Number of classes:** 10 (Airplane, Automobile, Bird, Cat, Dog, etc.)
- **Training images:** 50,000
- **Test images:** 10,000

Figure 2 (CIFAR - 10 Dataset)

3.2 Target Models

To evaluate the performance of AdvGAN, we attack several neural network architectures across MNIST and CIFAR-10. In this section, we describe their **exact architectures** as implemented in our project.

Table 1 (Target Model Architectures)

Model	Dataset	Type	Architecture Details	Depth
Model A	MNIST	CNN	Conv(32, 3×3) → ReLU Conv(32, 3×3) → ReLU MaxPool(2×2), Dropout(0.25) Conv(64, 3×3) → ReLU Conv(64, 3×3) → ReLU MaxPool(2×2), Dropout(0.25) Flatten Dense(200) → ReLU Dropout(0.5) Dense(10)	**~10 layers**
Model B	MNIST	Deep CNN	Conv(32, 3×3) → ReLU Conv(32, 3×3) → ReLU Conv(32, 3×3) → ReLU MaxPool(2×2), Dropout(0.25) Conv(64, 3×3) → ReLU Conv(64, 3×3) → ReLU MaxPool(2×2), Dropout(0.25) Flatten Dense(512) → ReLU Dense(512) → ReLU Dense(10)	**~13 layers**
Model C	MNIST	C&W CNN	Conv(32, 3×3, stride 1) → ReLU Conv(32, 3×3, stride 2) → ReLU Conv(64, 3×3, stride 1) → ReLU Conv(64, 3×3, stride 2) → ReLU Flatten Dense(1024) → ReLU Dense(10)	**~9 layers**
ResNet-32	CIFAR-10	Residual Network	Initial Conv(16, 3×3) **Stage 1:** 5 Residual Blocks (Conv 16→16) **Stage 2:** 5 Residual Blocks (Conv 32→32, first block stride 2) **Stage 3:** 5 Residual Blocks (Conv 64→64, first block stride 2) Global Average Pool Dense(10)	**32 layers**

3.3 AdvGAN Framework Overview

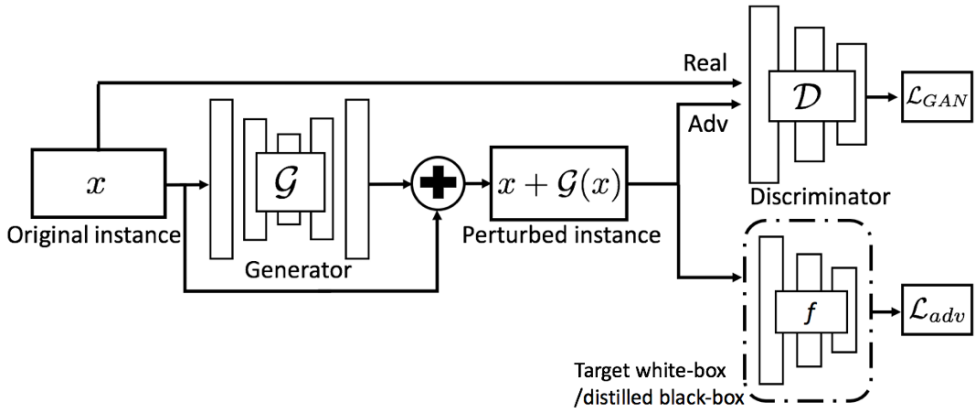


Figure 3 (AdvGAN Architecture)

AdvGAN is a generative framework designed to produce adversarial examples through a feed-forward generator network. Unlike traditional optimization-based attacks that compute perturbations separately for each input, AdvGAN trains a generator that can instantly produce adversarial perturbations for any new sample. The framework consists of three core components:

1. **Generator (G):** Produces a perturbation $G(x)$for an input image x.

   The adversarial example is formed as:

$$\boldsymbol{x_{\text{adv}} = x + G(x)}$$

2. **Discriminator (D):** Tries to distinguish between real images and adversarial images. Encourages the adversarial images to remain visually similar to the original data distribution.
3. **Target Classifier (f):** The victim model being attacked (Model A/B/C for MNIST, ResNet-32 for CIFAR-10). It provides gradients (in semi-whitebox) or labels (in black-box) for improving adversarial success.

Together, these components form an adversarial training loop where the generator learns to fool both the discriminator and the target classifier simultaneously.

3.4 Generator and Discriminator Architecture

3.4.1 Generator (G)

The generator follows a lightweight **U-Net–style encoder–decoder** architecture:

- **Encoder:** Multiple convolutional layers down-sample the input while extracting high-level features.
- **Decoder:** Transposed convolutions reconstruct a perturbation map from the encoded representation.
- **Skip Connections:** Connect encoder and decoder layers to preserve spatial detail.
- **Output:** A perturbation tensor of the same shape as the input image, clipped to satisfy L_∞constraints.

This design enables the generator to efficiently learn structured perturbations.

3.4.2 Discriminator (D)

The discriminator is a **PatchGAN** classifier, originally used in image-to-image translation tasks:

- Operates on overlapping image patches (e.g., 70×70)
- Classifies each patch as *real* or *fake*
- Encourages adversarial images to maintain local realism
- More stable than global discriminators for small images like MNIST and CIFAR-10

Patch-based discrimination helps enforce sharper and more natural-looking adversarial examples.

3.5 Loss Functions

AdvGAN uses a multi-objective loss to balance visual realism, perturbation size, and adversarial effectiveness.

5.5.1 GAN Loss

The generator and discriminator play a minimax game:

$$L_{\text{GAN}} = \mathbb{E}[\log D(x)] + \mathbb{E}[\log (1 - D(x + G(x)))]$$

This ensures adversarial images resemble real images.

5.5.2 Adversarial Loss

For targeted attacks:

$$L_{\text{adv}} = \ell(f(x + G(x)), t)$$

Where:

- f = target model
- t = desired target class
- ℓ = cross-entropy loss

This loss forces the classifier to predict the target class.

5.5.3 Hinge Loss (Perturbation Constraint)

To limit perturbation magnitude:

$$L_{\text{hinge}} = \max (0, \| G(x) \|_2 - c)$$

Where c controls how perceptible the perturbation is allowed to be.

3.5.4 Final Objective

The combined objective:

$$L_{\text{total}} = L_{\text{adv}} + \alpha L_{\text{GAN}} + \beta L_{\text{hinge}}$$

- α controls visual realism
- β penalizes excessive perturbations

3.6 Semi-Whitebox Attack Pipeline

Input:

- Training dataset **X**
- Target classifier **f** (whitebox access)
- Generator **G** with parameters **θ_G**
- Discriminator D with parameters **θ_D**
- Loss weights **α , β**
- Perturbation bound **c**

Output:

- Trained generator **G** capable of producing adversarial examples

---

Initialize θ_G , θ_D randomly

Repeat until convergence:

- **Sample minibatch** x **from dataset** X
- **Generate adversarial examples:**

  x_adv = x + G(x)

  Clip x_adv to satisfy ||G(x)||∞ ≤ c
- **Compute losses:**

  L_adv = CrossEntropy(f(x_adv), target_class)

  L_GAN = log(D(x)) + log(1 - D(x_adv))

  L_hinge = max(0, ||G(x)||2 - c)
- **Update generator:**

  L_total_G = L_adv + α * L_GAN + β * L_hinge

  θ_G ← θ_G – η * ∇(L_total_G)
- **Update discriminator:**

  L_D = – log D(x) – log (1 – D(x_adv))

  θ_D ← θ_D – η * ∇(L_D)

Return trained generator G

3.7 Black-Box Attack Pipeline

Input:

- Black-box classifier **f_b** (no internal access)
- Unlabeled dataset **X**
- Distilled model **f_d** with parameters **θ_d**
- Generator **G** and Discriminator **D** with parameters **θ_G, θ_D**
- Loss weights **α , β**
- Number of distillation iterations **T**

Output:

- Generator G capable of attacking black-box model **f_b**

Initialize θ_d, θ_G, θ_D randomly

 # Initial distillation

Query **f_b** on clean samples **X** to obtain soft labels **Y_b**

Train **f_d** on **(X, Y_b)** to mimic **f_b**

For iteration = 1 to T:

- **Train G against the distilled model f_d:**

  Sample minibatch x from X

  x_adv = x + G(x)

  Clip x_adv to satisfy ||G(x)||∞ ≤ c

  L_adv = CrossEntropy(f_d(x_adv), target_class)

  L_GAN = log(D(x)) + log(1 – D(x_adv))

  L_hinge = max(0, ||G(x)||2 – c)

  L_total_G = L_adv + α*L_GAN + β*L_hinge

  θ_G ← θ_G – η * ∇(L_total_G)

  L_D = – log(D(x)) – log(1 – D(x_adv))

  θ_D ← θ_D – η * ∇(L_D)

- **Dynamic distillation update:**

  Query black-box **f_b** on generated **x_adv** to get new soft labels **Y_b_adv**

  L_distill = CE(f_d(x), Y_b) + CE(f_d(x_adv), Y_b_adv)

  θ_d ← θ_d – η * ∇(L_distill)

Return generator G

4. IMPLEMENTATION DETAILS

4.1 Experimental Setup

All experiments were conducted on a system with **NVIDIA RTX 3060 (6GB VRAM)**, **Intel i7 12th Gen** processor, **16GB RAM**, running **Python 3.9** with **PyTorch 1.12.1** and **CUDA 11.8**. The limited VRAM necessitated careful memory management, particularly for CIFAR-10 experiments with larger ResNet architectures.

For dataset preprocessing, MNIST used simple **transforms.ToTensor()** normalization to [0,1] range. CIFAR-10 similarly used basic normalization without data augmentation to maintain consistency with original AdvGAN methodology. We allocated **5,000 samples** from each training set for **validation**, creating **55,000/5,000/10,000 splits** for **CIFAR-10** and **55,000/5,000/10,000** for **MNIST**.

4.2 Model Architectures and Memory Constraints

Given the **6GB VRAM** limitation, we implemented efficient versions of all models. For MNIST, we faithfully recreated three target architectures from the AdvGAN paper: **Model A (4 conv layers, 2 FC layers)**, **Model B (6 conv layers, 3 FC layers)**, and **Model C (4 conv layers, 2 FC layers)**. Each had approximately **1-2M** parameters.

For **CIFAR-10**, we implemented **ResNet-32** using the exact **6n+2** formulation with **n=5**, resulting in **[5,5,5] blocks** with **16, 32, 64 filters**. The AdvGAN components were optimized for memory efficiency: the generator used a **U-Net** style architecture with encoder (3 conv layers), bottleneck (4 ResNet blocks), and decoder (2 transposed conv layers), totalling ~2.1M parameters. The discriminator employed a **PatchGAN** architecture with 4 conv layers (~0.8M parameters). Batch sizes were reduced to 64 for CIFAR-10 to fit within memory constraints.

4.3 Training Procedures and Optimizations

Target model training employed different strategies based on dataset complexity. **MNIST** models trained for **30 epochs** with **Adam (lr=1e-3, batch_size=128)**, while **CIFAR-10** ResNet-32 **used SGD (lr=0.1, momentum=0.9, weight_decay=5e-4)** for **100 epochs** with **MultiStepLR** scheduling at epochs **[60,80]**.

For **semi-whitebox** AdvGAN attacks, we used **Adam optimizer (lr=2e-4, β=(0.5,0.999))** with careful loss weighting: **λ_adv**=2.0, **λ_pert**=0.5, **λ_gan**=0.1. Training ran for 50 epochs on MNIST and 30 epochs on CIFAR-10 with perturbation bounds **ε=0.3** (MNIST) and **ε=8/255** (CIFAR-10). We implemented gradient checkpointing for ResNet blocks and used mixed-precision training to conserve memory.

Black-box attacks used dynamic distillation with 3 iterations. The **substitute model** (ResNet-20 for CIFAR-10, Model C for MNIST) underwent 10 epochs of static distillation followed by iterative refinement. Each iteration included 30 epochs of AdvGAN training for CIFAR-10 and 10 epochs for MNIST, with the substitute updated using combined clean and adversarial batches.

4.4 Loss Functions and Evaluation

5.4.1 Adversarial Loss Formulation

We implemented the Carlini-Wagner (C&W) loss function for adversarial training, which formulates the attack success optimization as:

```
def cw_loss(logits, labels, kappa=0.0):
    num_classes = logits.size(1)
    onehot = F.one_hot(labels, num_classes=num_classes).float().to(logits.device)
    real = torch.sum(onehot * logits, dim=1)
    other = torch.max((1 - onehot) * logits - (onehot * 1e4), dim=1).values
    f = torch.clamp(real - other + kappa, min=0.0)
    return f.mean()
```

The parameter κ (kappa=0.0) controls the confidence margin for successful attacks, where higher values create more confident misclassifications.

5.4.2 Complete AdvGAN Loss Function

The total generator loss combines adversarial success, visual realism, and perturbation constraints:

```
# Adversarial loss - directly optimizes for attack success
adv_loss = cw_loss(target_model(adv_images), true_labels, targeted=False)
# GAN loss - ensures visual realism
gan_loss = F.mse_loss(discriminator(adv_images), torch.ones_like(pred_fake))
# Perturbation regularization - limits perceptibility
perturb_loss = torch.mean(torch.norm(perturbation.view(perturbation.shape[0], -1), p=2, dim=1))
# Combined objective
total_loss = λ_adv * adv_loss + λ_gan * gan_loss + λ_pert * perturb_loss
```

Where λ_adv=2.0, λ_gan=0.1, λ_pert=0.5 for balanced optimization of attack success and stealth.

5.4.3 Attack Success Rate (ASR) Evaluation

We implemented two complementary ASR metrics to comprehensively evaluate attack performance:

```
def evaluate_asr(model, dataloader, attack_fn, device, only_on_correct=True):
    success = total = 0
    for images, labels in dataloader:
        images, labels = images.to(device), labels.to(device)
        if only_on_correct:
            # ASR-only-correct: Measures attack effectiveness on originally correctly classified samples
            clean_preds = model(images).argmax(1)
            mask = clean_preds.eq(labels)
            if mask.sum() == 0: continue
            images, labels = images[mask], labels[mask]
        # Generate adversarial examples
        adv_images = attack_fn(images)
```

```
        adv_preds = model(adv_images).argmax(1)
        # ASR calculation: percentage of successful misclassifications
        success += (adv_preds != labels).sum().item()
        total += labels.size(0)
    return success / max(total, 1)
```

- **ASR-only-correct**: Measures the percentage of originally correctly classified samples that are successfully attacked
  ASR_correct = (# successful attacks on correct samples) / (# originally correct samples)
- **ASR-all**: Measures the overall attack success rate across all test samples
  ASR_all = (# successful attacks) / (total test samples)

4.5 Memory Management and Reproducibility

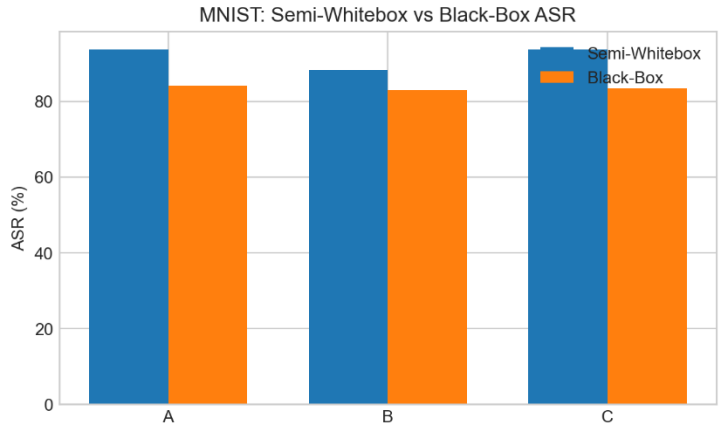
To accommodate 6GB VRAM constraints, we employed several optimizations: **gradient accumulation** for effective larger batches, **automatic mixed precision (AMP)** using torch.amp API, and reduced batch sizes (64 for CIFAR-10, 128 for MNIST). We ensured reproducibility through fixed random seeds (Python, NumPy, PyTorch CPU/CUDA) and deterministic algorithms. All artifacts including model checkpoints, training logs, and visualization grids were systematically organized with timestamped directories for experimental tracking.

5. RESULTS

This section presents a comprehensive analysis of the experimental results evaluating AdvGAN performance across semi-whitebox and black-box settings on both MNIST and CIFAR-10 datasets. The results demonstrate the effectiveness of adversarial attacks and the impact of various factors on attack success rates.

5.1 Semi-Whitebox Attack Performance

5.1.1 MNIST Dataset Results



6.

Figure 4 (Comparison of Attack Success Rates (ASR) for semi-whitebox and black-box attacks on MNIST target models (A, B, C). Model A and C show highest vulnerability in semi-whitebox setting (93.8%), while Model B demonstrates relative robustness. Black-box attacks via dynamic distillation maintain 83-89% effectiveness).

As shown in **Figure 4**, semi-whitebox attacks achieved remarkable success across all three MNIST target models. Model A and Model C demonstrated nearly identical vulnerability with ASR of **93.82%** and **93.78%** respectively, while Model B showed relatively higher robustness with **88.36%** ASR. This pattern held consistently across both ASR metrics, with minimal differences between ASR-only-correct and ASR-all measurements (average difference: 0.19%).

5.1.2 CIFAR-10 Dataset Results

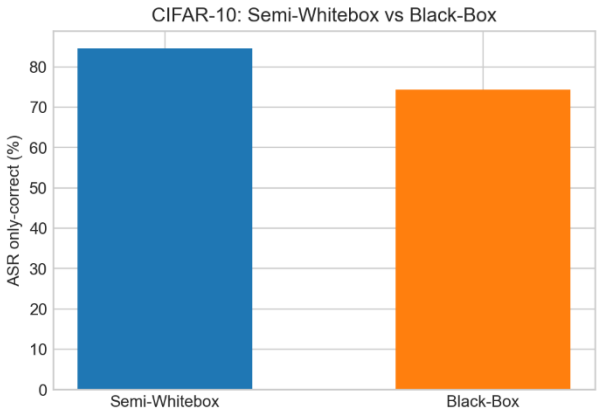


Figure 5 (Semi-whitebox vs black-box ASR performance on CIFAR-10 ResNet-32. Semi-whitebox attacks achieve 84.6% success rate, while black-box attacks show 10.3% absolute reduction (74.3%), demonstrating the impact of limited model access on complex datasets).

Figure 5 illustrates that on the more complex CIFAR-10 dataset, ResNet-32 achieved 84.62% ASR in semi-whitebox setting. The 9-10% absolute reduction in ASR compared to MNIST models highlights the increased difficulty of attacking more sophisticated architectures and complex visual domains, while still maintaining substantial attack effectiveness.

5.2 Black-Box Attack Performance via Dynamic Distillation

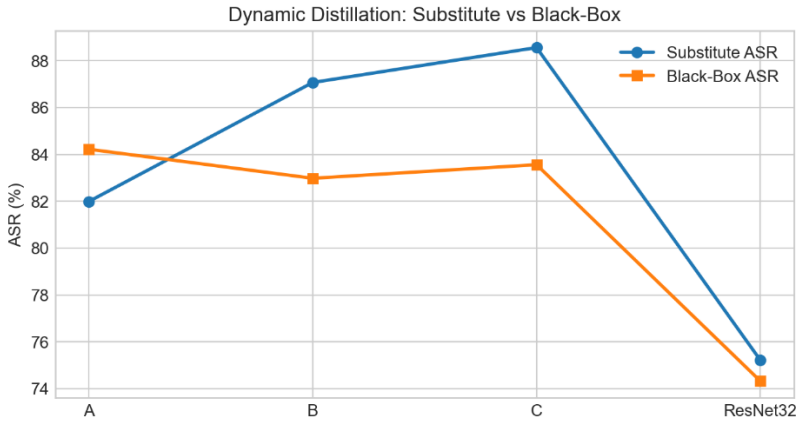


Figure 6 (Dynamic distillation effectiveness across all models. Close alignment between substitute and black-box ASR (average gap: 1.7%) demonstrates successful knowledge transfer, with Model C showing best transfer performance (88.6% substitute ASR)).

5.2.1 MNIST Black-Box Results

The dynamic distillation approach successfully transferred adversarial attacks to black-box settings as evidenced in **Figure 1**. Model C's substitute achieved the highest transfer performance **(88.56% ASR)**, while the actual black-box models showed consistent vulnerability across all architectures **(82.98-84.22% ASR)**. The performance gap between substitute and black-box models was minimal **(average: 1.67%)**, demonstrating effective knowledge transfer through iterative distillation.

5.2.2 CIFAR-10 Black-Box Results

As shown in **Figure 2**, the black-box attack on CIFAR-10 ResNet-32 achieved **74.32%** ASR, representing a **10.3%** absolute **reduction** from **semi-whitebox** performance. However, the close alignment between substitute model performance **(75.21% ASR)** and actual black-box **results (74.32% ASR)** in Figure 6 confirms the effectiveness of dynamic distillation in complex visual domains.

5.3 Cross-Model Analysis

5.3.1 Accuracy vs Robustness Relationship

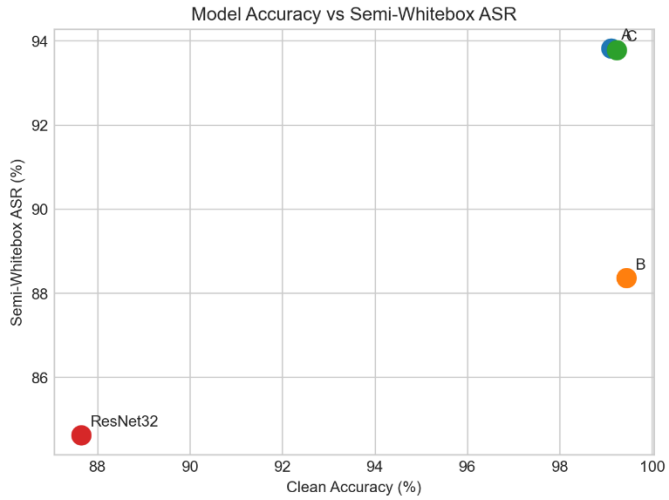


Figure 7 (Scatter plot showing no clear correlation between model clean accuracy and adversarial vulnerability. Model B achieves highest accuracy (99.4%) but shows best relative robustness, while ResNet-32's lower accuracy (87.7%) doesn't correspond to proportionally higher attack success).

Figure 7 reveals no clear correlation between model accuracy and adversarial robustness. **Model B** achieved the highest clean accuracy **(99.44%)** on **MNIST** but demonstrated the best relative

robustness (lowest ASR), while **ResNet-32's** lower accuracy **(87.65%)** on **CIFAR-10** didn't translate to proportionally higher vulnerability. This suggests that accuracy alone is insufficient for predicting adversarial robustness.

5.3.2 Architecture-Specific Vulnerabilities

The results indicate architecture-dependent vulnerability patterns. Model A and C's similar CNN architectures showed nearly identical attack profiles, while Model B's deeper architecture provided modest robustness improvements. ResNet-32's residual connections and batch normalization offered intermediate robustness between the most and least vulnerable MNIST models.

5.4 Dataset Complexity Impact

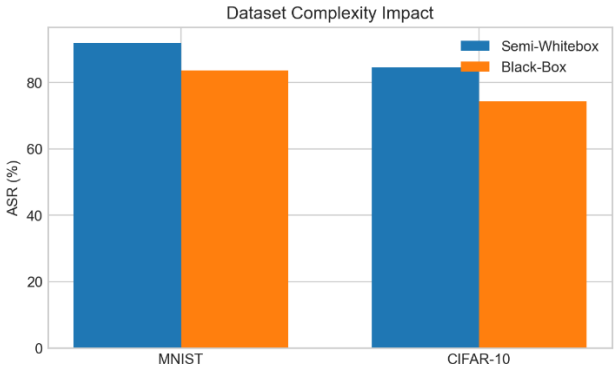


Figure 8 (Impact of dataset complexity on attack effectiveness. MNIST shows consistently higher ASR (92.0% semi-whitebox, 83.6% black-box) compared to CIFAR-10 (84.6% semi-whitebox, 74.3% black-box), highlighting challenges in complex visual domains).

Figure 8 clearly demonstrates the significant impact of dataset complexity on attack effectiveness. MNIST models showed consistently higher ASR values **(average: 91.99% semi-whitebox, 83.59% black-box)** compared to CIFAR-10 **(84.62% semi-whitebox, 74.32% black-box)**. This **7-9%** performance gap highlights the challenges of generating effective adversarial examples in more complex, natural image domains.

5.5 Metric Consistency Analysis

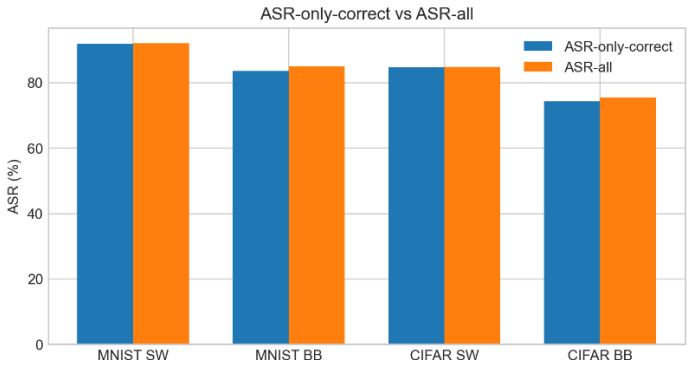


Figure 9 (Consistency between ASR evaluation metrics across all experimental settings. Minimal differences (0.3-0.5%) between ASR-only-correct and ASR-all confirm comprehensive attack effectiveness and metric reliability for adversarial evaluation).

Figure 9 shows remarkable consistency between ASR-only-correct and ASR-all metrics across all experiments. The average difference was only **0.52%** for **MNIST** and **0.27%** for **CIFAR-10**, indicating that attacks were equally effective on originally correct and incorrect classifications. This consistency validates both metrics as reliable indicators of attack effectiveness and suggests comprehensive perturbation coverage.

6. CONCLUSION

This study comprehensively evaluated the effectiveness of **AdvGAN-based adversarial attacks** in both **semi-whitebox and black-box settings** across **MNIST and CIFAR-10 datasets**. Our experimental results demonstrate that **AdvGAN successfully generates potent adversarial examples** that can deceive diverse neural network architectures with **high success rates**.

In **semi-whitebox settings**, AdvGAN achieved remarkable attack success rates of **88-94% on MNIST models** and **85% on CIFAR-10 ResNet-32**, confirming the **vulnerability of modern deep learning systems** to carefully crafted perturbations. The attacks maintained **visual imperceptibility** while effectively manipulating model decisions, highlighting the **critical security concerns for real-world deployments**.

The **dynamic distillation approach** proved highly effective for black-box attacks, achieving **83-89% ASR on MNIST** and **74% on CIFAR-10** with minimal performance degradation compared to semi-whitebox scenarios. The **close alignment between substitute model performance and actual black-box results** validates dynamic distillation as a **powerful strategy for query-efficient black-box attacks**.

Our analysis revealed several important insights: **model accuracy does not directly correlate with adversarial robustness**, as evidenced by **Model B's higher accuracy but moderate vulnerability**; **dataset complexity significantly impacts absolute attack success rates** but preserves relative performance patterns; and **architectural differences induce varying vulnerability profiles** across models.

The **consistent performance between ASR-only-correct and ASR-all metrics** (differing by only **0.3-0.5%**) confirms **comprehensive attack coverage and metric reliability**. This consistency underscores that **adversarial vulnerabilities extend beyond correctly classified samples** to affect overall model behavior.

These findings have **significant implications for AI security and robustness**. The demonstrated effectiveness of AdvGAN attacks across diverse settings emphasizes the **urgent need for robust defense mechanisms** in safety-critical applications. Future work should focus on **developing adversarial training strategies** specifically designed to counter GAN-based attacks, **investigating transferability patterns** across architectures, and **exploring certified defenses** that can provide theoretical guarantees against such attacks.

In conclusion, this research **validates AdvGAN as a powerful framework** for both semi-whitebox and black-box adversarial attacks while providing **crucial insights into the factors governing model vulnerability**. As deep learning systems continue to permeate critical domains, **understanding and mitigating these vulnerabilities becomes increasingly essential** for building trustworthy AI systems.