AdversaGuard: A Distributed Data Poisoning Benchmark for Parallel AI

Yulia Kumar, Solomon Thomas, Dejaun Gayle, Dov Kruger, and J. Jenny Li
Kean University, Union, NJ, USA; Rutgers University, Piscataway, NJ, USA

Abstract

AdversaGuard is an HPC-oriented framework for distributed data poisoning (DDP) that benchmarks adversarial robustness and training efficiency across parallel paradigms. We run eight configurations—seven distributed systems plus a non-distributed baseline—over three food-domain datasets and 4 models (from compact CNNs to large transformers). The suite implements 8 attacks (FGSM, PGD, DeepFool, One-Pixel, Universal, Carlini–Wagner, Trojan, Boundary) & analyzes how data, model, & hybrid parallelism affect scalability, memory use, and vulnerability. A companion app enables live testing and reproducible demos. Observed results show that some architectures can tolerate poisoning, and larger models are more susceptible at comparable budgets, underscoring the need for adaptive, scalable, parallelism-aware defenses.

Figure 1. AdversaGuard Framework at-a-Glance.

Research Questions: RQ1: Which distributed ML frameworks and parallelism methods are most vulnerable to poisoning in HPC settings? RQ2: How can malicious nodes be leveraged to amplify attacks while minimizing detection risk? RQ3: How does the choice of data/model/hybrid parallelism affect attack severity and detectability?

Computational Environment & AdversaGuard Efficiency Index (AEI)

Component	Specification
CPU Model	2× Intel(R) Xeon(R) E5-2680 v4 @ 2.40 GHz
Total Cores	28 (56 with Hyper-Threading)
Sockets	2
Architecture	x86_64
L3 Cache	35,840 KB
NUMA Nodes	2

$$\mathrm{AEI} = w_1\,\mathrm{CA} + w_2\,(1-\mathrm{ASR}) - \sum_{i=3}^{11} w_i\,\mathrm{Cost}_i, \qquad (1)$$

where CA is the clean accuracy, ASR is the attack success rate. The cost terms Cost*i* penalize perturbation size (e.g., PR or ℓ2 norm), model parameters (*N*params), computational latency (*T*CPU,*T*GPU), and training time (*T*train). Weights *wi* are user-tunable to prioritize specific aspects.

Food-Related Datasets

Figure 2. Tiny fisheye and Fungi datasets.

App Demo

References

Materials and Methods

Our technical approach involves applying each of the 8 attack types to every model across all available targeted distributed AI frameworks. The distributed methodology follows a 7-step process starting from initialization and data preparation to full attack generation, model training, and results evaluation.

Step 1: HPC and Distributed Initialization (Initialize HPC, Initialize distributed frameworks)

Step 2: Data Preparation (For each dataset: Split into train/validation/test; Resize, normalize, augment, etc.)

Step 3: Attack Generation Functions (For each attack: Define PoisonData function to modify dataset; if multiprocessing, partition and apply PoisonData)

Step 4: Nested Training Loop (For each framework, for each model, for each dataset, apply each attack)

Step 5: Poisoned Data Creation (Apply PoisonData to create poisoned dataset)

Step 6: Model Training (Initialize and distribute model;Train, track metrics like accuracy, ASR, memory, and time)

Step 7: Save logs for each combination, Compute metrics, create Visualizations

Figure 3. Distributed Methodology.

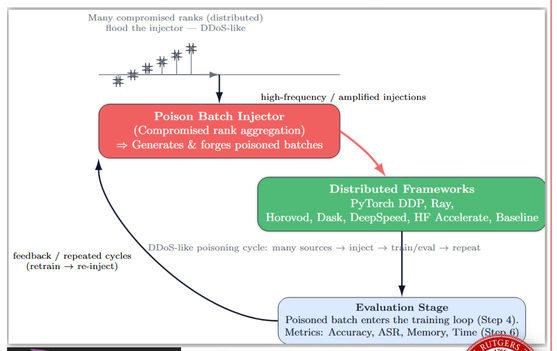

Figure 4. Batch Poisoning.

Results

To complement the benchmark, we developed the AdversaGuard interactive web application for live demonstration and testing. The system architecture is depicted in Fig. 5. App Workflow:(1) Start: A user selects a target model, an input image, an attack method, and its strength (e.g., ϵ).(2) Request: The client UI sends these selections to the backend.(3) Processing: The backend's attack engine generates the adversarial image and evaluates the model response, calculating metrics such as prediction confidence and attack success time.(4) Response: The backend returns the adversarial image (base64), predictions, and performance metrics.(5) End: The UI renders side-by-side original vs. adversarial images, with tables and rankings.

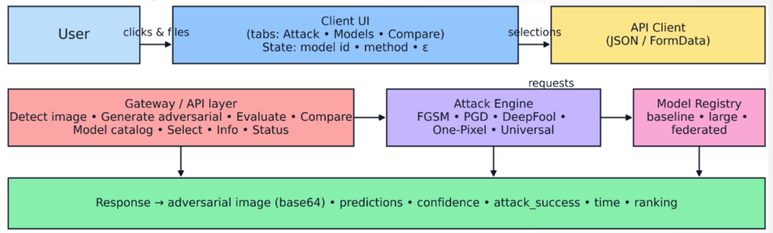

Figure 5: AdversaGuard app Architecture..

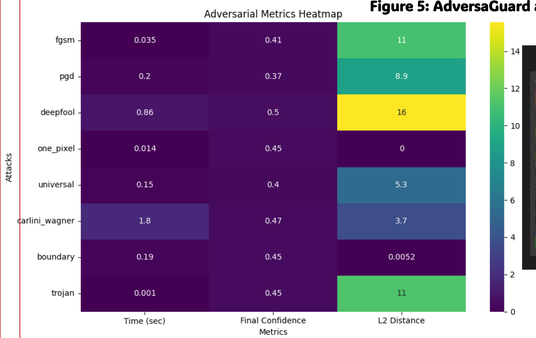

Figure 6: Effect of Data Poisoning on ResNet50.

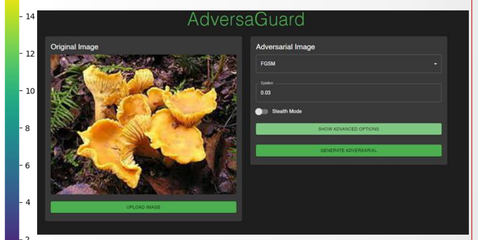

Figure 7: GUI of the AdversaGuard app.

Key Findings

(1) Batch Poisoning and the Impact of Parallelism. The choice of a parallel strategy directly influences attack effectiveness. In data-parallel training, gradient averaging can dilute or mask small-budget perturbations, making certain attacks less effective. To study this, we implement a Batch Poisoning strategy (Fig. 4), where a single rank periodically generates a poisoned batch injected into the training cycle.
(2) Scale vs. Susceptibility. Larger models (e.g., ViT), despite higher clean accuracy, often exhibit greater vulnerability to poisoning compared to smaller convolutional models at similar attack budgets. This suggests increased capacity can enlarge the attack surface, motivating capacity-aware defenses.
(3) Cost–Robustness Trade-offs. Results are summarized in the ResNet-50 metrics heatmap (Fig. 6), highlighting trade-offs between attacks. For instance, DeepFool achieves high-confidence misclassification with a small ℓ2 distance, while Boundary is computationally expensive yet highly effective. Our AEI method surfaces these trade-offs by integrating performance, efficiency, and robustness into a single score.