CHECKLIST

Checklist: Building Secure ML Models with Adversarial Training

A practical checklist for evaluating, implementing, and validating adversarial training for production machine learning systems.

Checklist: Building Secure ML Models with Adversarial Training

Use this checklist to decide whether adversarial training is appropriate for your model, implement it safely, and validate that it actually improves resilience in production.

1) Confirm the threat model

- [ ] Define the model?s role in production: classification, ranking, anomaly detection, decision support, or other.
- [ ] Identify the most likely adversary behavior:
- [ ] Small feature/value manipulation
- [ ] Text substitutions or token changes
- [ ] Image perturbations or corruption
- [ ] Missing-field, reorder, or formatting abuse
- [ ] Other: _____
- [ ] Specify what the adversary can and cannot change.
- [ ] Document the acceptable perturbation bounds.
- [ ] Confirm the perturbation assumptions are realistic for the deployment environment.

- [ ] Verify adversarial training is the right control for this threat, not a substitute for data hygiene, runtime detection, or secure deployment.

2) Assess fit before training

- [ ] Determine whether the model is exposed to user-controlled or externally sourced inputs.
- [ ] Check whether small manipulations could change model outputs in meaningful ways.
- [ ] Identify dependencies on upstream data quality and feature provenance.
- [ ] Estimate the training overhead:
- [ ] Extra compute for adversarial sample generation
- [ ] Longer training time
- [ ] Additional validation effort
- [ ] Decide whether a small clean-accuracy trade-off is acceptable.
- [ ] Confirm the team can re-run robustness testing after retraining or feature changes.

3) Choose a training approach

- [ ] Select the adversarial training pattern that matches the use case:
- [ ] Offline adversarial augmentation
- [ ] Minimax optimization
- [ ] Curriculum-style hardening
- [ ] Hybrid with standard augmentation or denoising
- [ ] Ensure the perturbation generator matches the model type:
- [ ] Vision
- [ ] NLP
- [ ] Tabular
- [ ] Other: _____
- [ ] Keep perturbations within the chosen bounds.

- [ ] Avoid assuming one robustness method generalizes across all attack types.
- [ ] Define what a successful hardening result should look like before training starts.

4) Prepare the training data and pipeline

- [ ] Confirm the base dataset is clean enough for training.
- [ ] Remove or flag obviously corrupted or invalid records.
- [ ] Check label quality and labeling consistency.
- [ ] Validate feature provenance where applicable.
- [ ] Ensure adversarial samples are generated from the current model state or a clearly defined attack procedure.
- [ ] Mix clean and adversarial examples in a controlled way.
- [ ] Track the ratio of clean to adversarial samples.
- [ ] Version the dataset, attack settings, and training configuration.

5) Validate robustness correctly

- [ ] Evaluate on a clean validation set.
- [ ] Evaluate on adversarial or worst-case test sets.
- [ ] Use attacks and perturbations that reflect the documented threat model.
- [ ] Compare performance against a baseline model trained without adversarial examples.
- [ ] Measure both:
- [ ] Clean accuracy or task quality
- [ ] Robustness under attack
- [ ] Check whether robustness gains are material, not just statistically minor.
- [ ] Test for regressions across key segments, classes, or feature groups.
- [ ] Confirm the model degrades more slowly under perturbation than the baseline.

6) Review operational impact

- [ ] Estimate training time increase.
- [ ] Estimate inference latency impact, if any.
- [ ] Identify memory or compute constraints.
- [ ] Confirm the robustness gain justifies the cost.
- [ ] Decide whether the model can tolerate slight clean-accuracy loss.
- [ ] Assess whether retraining frequency will need to change.
- [ ] Document any trade-offs accepted by the business or security owner.

7) Add runtime controls

- [ ] Implement input validation and schema checks.
- [ ] Add rate limiting where the model is exposed through an API.
- [ ] Monitor for unusual input patterns or distribution drift.
- [ ] Log rejected, suspicious, or out-of-policy requests.
- [ ] Consider adversarial or anomaly detection at inference time.
- [ ] Protect the model endpoint with secure deployment practices.
- [ ] Restrict access to model artifacts, training data, and inference services.
- [ ] Define a rollback path if robustness degrades after deployment.

8) Verify production readiness

- [ ] The threat model is documented and approved.
- [ ] The perturbation space matches realistic abuse cases.
- [ ] Robustness tests passed on representative clean and adversarial data.
- [ ] Clean-accuracy impact is understood and accepted.

- [ ] Runtime monitoring is in place.
- [ ] Secure deployment and access controls are in place.
- [ ] Retraining triggers are defined.
- [ ] Ownership for ongoing review is assigned.

9) Watch for common failure modes

- [ ] The threat model is too vague.
- [ ] The perturbations are unrealistic for the environment.
- [ ] The team treats adversarial training as a complete security solution.
- [ ] Validation only checks clean accuracy.
- [ ] Robustness is not re-tested after feature or data changes.
- [ ] Runtime controls are missing.
- [ ] The model is hardened against one attack class but remains exposed to others.

10) Go-live decision

Before release, answer these questions:

- [ ] Can we explain what attacks the model is hardened against?
- [ ] Can we show improved behavior under those attacks?
- [ ] Is the remaining risk acceptable?
- [ ] Are monitoring and response controls ready?
- [ ] Do we have a plan to revisit robustness after changes?

Release decision

- [ ] Proceed to production
- [ ] Proceed with additional controls first

- [ ] Do not deploy until the threat model and validation are improved

Quick summary

Adversarial training is most useful when:

- [ ] The model faces realistic input manipulation
- [ ] The perturbation space is clearly defined
- [ ] The team can validate robustness beyond clean accuracy
- [ ] The cost and accuracy trade-offs are acceptable
- [ ] Runtime protections are part of the overall design

If you cannot check most of these boxes, adversarial training is probably not ready for production use.