CHECKLIST

Checklist: Securing ML Models with Adversarial Attack Detection

A practical checklist for evaluating, deploying, and operating adversarial attack detection around machine learning inference workflows.

Checklist: Securing ML Models with Adversarial Attack Detection

Use this checklist to assess whether your machine learning system has enough protection against adversarial inputs before production use.

1) Confirm the risk is real for this model

- [ ] Identify what the model influences: access control, fraud scoring, malware classification, content moderation, KYC, or automated remediation.
- [ ] Determine whether external users can repeatedly query the model.
- [ ] Verify whether predictions, labels, or confidence scores reveal thresholds or decision boundaries.
- [ ] Assess whether small input changes can materially affect outcomes.
- [ ] Decide whether the model is valuable enough to attack and iterable enough for probing.

2) Define what ?adversarial? means in your environment

- [ ] Separate adversarial inputs from malformed, invalid, and out-of-distribution inputs.
- [ ] Document the attack patterns you care about most.

- [ ] List likely input types: text, images, documents, files, events, or feature vectors.
- [ ] Identify whether the attacker can manipulate raw input, metadata, timing, or request frequency.
- [ ] Write down the expected response when suspicious input is detected.

3) Put basic validation in front of the model

- [ ] Enforce schema checks for required fields, types, and formats.
- [ ] Validate ranges, lengths, and allowed character sets.
- [ ] Canonicalize inputs before scoring to reduce encoding tricks.
- [ ] Reject impossible or contradictory values early.
- [ ] Log validation failures separately from adversarial detections.

4) Add multiple detection signals

- [ ] Use input abnormality checks for improbable values and feature combinations.
- [ ] Add confidence or margin checks for unstable predictions.
- [ ] Measure sensitivity to small perturbations where appropriate.
- [ ] Compare embeddings or feature-space distances against known-good traffic.
- [ ] Monitor request behavior for repeated queries, bursts, or boundary probing.
- [ ] Combine model-side and behavioral signals instead of relying on one detector.

5) Design the response path before deployment

- [ ] Define actions for each risk level: allow, flag, rate-limit, challenge, quarantine, or reject.
- [ ] Ensure each action has a clear operational owner.
- [ ] Route high-risk requests to human review when the cost of false positives is acceptable.

- [ ] Provide a safe fallback for cases where inference should not proceed.
- [ ] Avoid alerts that do not change behavior.

6) Calibrate thresholds carefully

- [ ] Choose thresholds based on the action taken, not just detector score.
- [ ] Measure false positives and false negatives on representative traffic.
- [ ] Test how thresholds behave under traffic spikes and new client integrations.
- [ ] Confirm whether a lower threshold is acceptable for review workflows.
- [ ] Confirm whether user-facing rejection requires stricter precision.

7) Log enough to investigate later

- [ ] Record request metadata, detector score, and decision outcome.
- [ ] Keep enough context to distinguish attack traffic from drift or upstream data issues.
- [ ] Store the reason a request was flagged.
- [ ] Preserve samples for investigation subject to privacy and retention rules.
- [ ] Make sure logs are searchable by time, source, model version, and action taken.

8) Validate against realistic attack scenarios

- [ ] Test near-boundary perturbations that shift predictions without breaking syntax.
- [ ] Test repeated near-duplicate submissions.
- [ ] Test bursts from a narrow origin set or unusual user-agent patterns.
- [ ] Test feature combinations that are plausible but uncommon.
- [ ] Test whether the detector still works after preprocessing or model updates.

9) Watch for drift and operational noise

- [ ] Review adversarial alerts alongside model drift metrics.
- [ ] Compare alert spikes with product releases, new integrations, and data pipeline changes.
- [ ] Check whether upstream feed corruption could explain the anomaly.
- [ ] Distinguish true probing from benign edge-case behavior.
- [ ] Revisit thresholds after major changes to data, model, or traffic patterns.

10) Prepare governance and ownership

- [ ] Assign ownership across ML, security, and operations.
- [ ] Document escalation paths for suspicious inference attempts.
- [ ] Define when a flagged event becomes a security incident.
- [ ] Review privacy, retention, and access controls for detector outputs.
- [ ] Include adversarial detection in your production readiness checklist.

11) Verify the control is actually useful

- [ ] Confirm that flagged requests trigger a meaningful change in handling.
- [ ] Measure whether the detector reduces attacker probing success.
- [ ] Track the volume of suspicious events over time.
- [ ] Review whether analyst workload remains manageable.
- [ ] Reassess whether the control should be tightened, relaxed, or layered with another safeguard.

Production readiness summary

Before going live, you should be able to answer yes to the following:

- [ ] Do we know which model outputs are most attractive to attackers?

- [ ] Can we detect suspicious inputs using more than one signal?
- [ ] Do flagged requests change behavior in a concrete way?
- [ ] Can we investigate alerts with enough context to make a decision?
- [ ] Have we tested the detector against realistic perturbations and probing patterns?
- [ ] Do we review alerts together with drift, data quality, and pipeline monitoring?
- [ ] Do we know who owns the response when the detector fires?

Quick rule of thumb

If the model is both valuable to attack and iterable by an adversary, adversarial attack detection should be part of the design.

If you cannot explain what happens after a request is flagged, the detector is not ready for production.