



T01

AI SECURITY

THREAT

Data poisoning

Training or fine-tuning code ingests data nobody vetted: user feedback, scraped pages, partner files, open datasets. A few crafted records can plant a backdoor or skew decisions.

WHERE IT SHOWS UP

Data loaders, feedback pipelines, fine-tuning jobs, RAG ingestion

ATTACKER NEEDS

Write access to any data source you train on

FIX WITH

AIC-01, AIC-02, AIC-04

VULNERABLE PATTERN · SPOT IT IN REVIEW, NEVER SHIP IT

VULNERABLE · PYTHON 3.12

vuln/py/t01.py

```
# VULNERABLE (T01): anything can reach training
import json
import urllib.request

def nightly_finetune(trainer, db) -> None:
    # user feedback, unreviewed, straight in
    rows = db.all("SELECT text, label FROM feedback")
    url = "https://partner.example/extra.jsonl"
    with urllib.request.urlopen(url) as r: # no hash
        rows += [json.loads(line) for line in r]
    trainer.fit(rows) # no provenance record
    trainer.deploy() # no golden-set gate
```

VULNERABLE · TYPESCRIPT 5 · NODE 22

vuln/ts/t01.ts

```
// VULNERABLE (T01): anything can reach training
export async function nightlyFinetune(trainer: Trainer,
db: Db) {
    // user feedback, unreviewed, straight in
    const rows = await db.all("SELECT text, label FROM fb");
    const res = await fetch("https://partner.example/x");
    for (const line of (await res.text()).split("\n")) {
        if (line) rows.push(JSON.parse(line)); // no hash
    }
    await trainer.fit(rows); // no provenance record
    await trainer.deploy(); // no golden-set gate
}
```

Code review checklist

- 01 Every data source in the loader is on an allow-list
- 02 Datasets are pinned by hash, not fetched live at train time
- 03 User-generated content is quarantined and reviewed first
- 04 Deploy sits behind a golden-set promotion gate

Tests that prove the fix

- 01 One changed byte in a dataset stops the run
- 02 A dataset from an unlisted host is rejected
- 03 A model trained on a seeded poisoned set fails the gate

THE FIX Record provenance and a SHA-256 hash for every dataset, verify it before each run, and gate promotion on a protected golden set.



T02

Evasion

AI SECURITY

THREAT

A predictive model makes a hard decision on one threshold. Small, deliberate changes to inputs such as amount, timing or wording push a bad case to the other side.

WHERE IT SHOWS UP

Fraud, credit, moderation and anomaly scoring services

ATTACKER NEEDS

Ability to submit inputs and watch outcomes

FIX WITH

AIC-05, AIC-06, AIC-15

VULNERABLE PATTERN · SPOT IT IN REVIEW, NEVER SHIP IT

VULNERABLE · PYTHON 3.12

vuln/py/t02.py

```
# VULNERABLE (T02): one score, one threshold, no checks
def screen_payment(model, tx: dict) -> str:
    x = [tx["amount"], tx["hour"], tx["merchant_risk"]]
    score = model.predict_proba(x)[0][1]
    if score < 0.5:      # a small nudge flips this
        return "approve" # no human near the line
    return "block"

# Never tested against crafted inputs, and nothing
# checks that features sit inside the training range.
```

VULNERABLE · TYPESCRIPT 5 · NODE 22

vuln/ts/t02.ts

```
// VULNERABLE (T02): one score, one threshold, no checks
export function screenPayment(model: Model, tx: Tx) {
    const x = [tx.amount, tx.hour, tx.merchantRisk];
    const score = model.predictProba(x)[0][1];
    return score < 0.5 ? "approve" : "block";
}

// A small nudge to amount or hour flips the result.
// Never tested against crafted inputs, no human near
// the threshold, no range check on the features.
```

Code review checklist

- 01 Decisions near the threshold go to a person
- 02 Features are range-checked against the training data
- 03 A robustness test with a written budget runs in CI
- 04 Decision logs keep inputs so attacks can be traced

Tests that prove the fix

- 01 Flip rate under the agreed budget stays below the limit
- 02 Out-of-range features are rejected or flagged
- 03 Scores in the review band route to a human queue

THE FIX Test robustness to a set perturbation budget before release and send borderline scores to human review.

**T03**

AI SECURITY

THREAT

Membership inference

An overfit model returns precise confidences. Comparing the confidence on a known record with typical values tells an attacker whether that person was in the training set.

WHERE IT SHOWS UP

Prediction APIs over personal data, especially health and finance

ATTACKER NEEDS

Query access and a candidate record

FIX WITH

AIC-07, AIC-09, AIC-08

VULNERABLE PATTERN · SPOT IT IN REVIEW, NEVER SHIP IT

VULNERABLE · PYTHON 3.12

vuln/py/t03.py

```
# VULNERABLE (T03): full confidences, overfit model
def predict(model, patient: dict) -> dict:
    probs = model.predict_proba([to_features(patient)])[0]
    return {"classes": list(model.classes_),
            "probabilities": [float(p) for p in probs]}
# Trained to 100% training accuracy, no privacy test.
# A near-1.0 confidence tells an attacker the record
# was probably in the training set.
```

VULNERABLE · TYPESCRIPT 5 · NODE 22

vuln/ts/t03.ts

```
// VULNERABLE (T03): full confidences, overfit model
export function predict(model: Model, patient: Patient) {
    const x = features(patient);
    const probs = model.predictProba([x])[0];
    return { classes: model.classes, probabilities: probs };
}
// Trained to 100% training accuracy, no privacy test.
// A near-1.0 confidence tells an attacker the record
// was probably in the training set.
```

Code review checklist

- 01 The API returns a label or band, not probabilities
- 02 The gap between train and holdout accuracy is bounded
- 03 A membership inference test runs before release
- 04 The DPIA sets the leakage threshold

Tests that prove the fix

- 01 Attack AUC on train against holdout is within the DPIA limit
- 02 The response schema rejects extra fields
- 03 Rate limits stop bulk probing

THE FIX Return coarse outputs only, and fail the release when membership AUC exceeds the DPIA threshold.

T04

AI SECURITY

THREAT

Model exfiltration

An endpoint lets anyone query without limits and returns rich outputs such as logprobs. Enough queries let an attacker train a copy of your model.

WHERE IT SHOWS UP

Public or partner inference APIs and LLM proxies

ATTACKER NEEDS

API access, often just a free account

FIX WITH

AIC-08, AIC-07

VULNERABLE PATTERN · SPOT IT IN REVIEW, NEVER SHIP IT

VULNERABLE · PYTHON 3.12

vuln/py/t04.py

```
# VULNERABLE (T04): an open, unmetered oracle
def complete(request: dict, client) -> dict:
    # no auth, no quota, no per-key usage alerts
    return client.generate(prompt=request["prompt"],
                           logprobs=True,
                           top_logprobs=20)

# Every token's alternatives go back to the caller:
# ideal training data for a copy of your model.
```

VULNERABLE · TYPESCRIPT 5 · NODE 22

vuln/ts/t04.ts

```
// VULNERABLE (T04): an open, unmetered oracle
export async function complete(req: Request) {
    // no auth, no quota, no per-key usage alerts
    const { prompt } = await req.json();
    const out = await llm.generate({ prompt,
    logprobs: true, topLogprobs: 20 });
    return Response.json(out); // alternatives included
}
```

Code review checklist

- 01 Every model route requires authentication
- 02 Per-identity quotas and rate limits are set
- 03 logprobs and top-k alternatives are off for outside callers
- 04 Alerts fire on high-volume or systematic querying

Tests that prove the fix

- 01 Requests beyond the burst are refused per identity
- 02 A sweep of distinct queries raises an alert
- 03 External responses contain no logprobs

THE FIX Authenticate every call, throttle per identity, alert on sweeps and return the minimum detail.

**T05**

AI SECURITY

THREAT

Model inversion

The model returns exact per-class scores, so an attacker can keep adjusting an input to raise one person's score and rebuild features of that person.

WHERE IT SHOWS UP

Face, voice and identity models; any model with one class per person

ATTACKER NEEDS

Query access to scores

FIX WITH

AIC-07, AIC-09, AIC-08

VULNERABLE PATTERN · SPOT IT IN REVIEW, NEVER SHIP IT

VULNERABLE · PYTHON 3.12

vuln/py/t05.py

```
# VULNERABLE (T05): per-person confidence on request
def who_is_this(face_model, image: bytes) -> dict:
    scores = face_model.scores(image) # one per person
    return {n: round(s, 6) for n, s in scores.items()}
# The caller sees how close every guess is. An
# attacker can climb these scores to rebuild a face.
```

VULNERABLE · TYPESCRIPT 5 · NODE 22

vuln/ts/t05.ts

```
// VULNERABLE (T05): per-person confidence on request
export function whoIsThis(faceModel: Model, img: Buffer) {
    const scores: Record<string, number> =
        faceModel.scores(img); // one score per person
    return scores; // every name with its exact score
}
// An attacker can climb these scores to rebuild a face.
```

Code review checklist

- 01 Scores are banded, or only the top label is returned
- 02 Per-person classes are hidden from untrusted callers
- 03 Privacy-preserving training is considered for sensitive sets
- 04 Query volume per identity is limited

Tests that prove the fix

- 01 Responses carry only a label and a band
- 02 An inversion attempt on a held-out identity fails
- 03 The throttle blocks iterative querying

THE FIX Coarsen outputs and test for inversion before release; use privacy-preserving training for sensitive data.

**T06**

AI SECURITY

THREAT

Direct model poisoning

A model or adapter is swapped or tampered with on its way to production. Pickle files and remote-code loaders run whatever the file's author wrote.

WHERE IT SHOWS UP

Model download, registry, build and deployment steps

ATTACKER NEEDS

Write access to a model source, registry or bucket, or a malicious hub upload

FIX WITH

AIC-03, AIC-04, AIC-02

VULNERABLE PATTERN · SPOT IT IN REVIEW, NEVER SHIP IT

VULNERABLE · PYTHON 3.12

vuln/py/t06.py

```
# VULNERABLE (T06): untrusted code runs at load time
import pickle

from transformers import AutoModel

model = AutoModel.from_pretrained(
    "some-user/fast-model",
    trust_remote_code=True) # runs the repo's code

with open("downloads/adapter.pkl", "rb") as f:
    adapter = pickle.load(f) # can run code on load
```

VULNERABLE · TYPESCRIPT 5 · NODE 22

vuln/ts/t06.ts

```
// VULNERABLE (T06): model loaded from anywhere
import { writeFile } from "node:fs/promises";

export async function updateModel(url: string) {
  const res = await fetch(url); // any host
  const bytes = Buffer.from(await res.arrayBuffer());
  await writeFile("/models/current.onnx", bytes);
  // no signature, no hash, no registry: whoever can
  // change the URL or the response owns the model
  await runtime.reload("/models/current.onnx");
}
```

Code review checklist

- 01 No pickle or joblib loads of files from outside
- 02 `trust_remote_code` is never `True`
- 03 Artefacts are signed in the build and verified at load
- 04 Models load only from the internal registry

Tests that prove the fix

- 01 A tampered artefact fails signature verification
- 02 A .pkl file is refused before any parsing
- 03 A load from an unregistered URL is refused

THE FIX Use safetensors or ONNX, sign artefacts in the build, verify them at load and keep an AI-BOM.

T07

AI SECURITY

THREAT

Direct model theft

Weights leave your control directly: a public bucket, a broad IAM role, a model shipped inside a client app, or a copy on a laptop.

WHERE IT SHOWS UP	ATTACKER NEEDS	FIX WITH
Object storage, model registries, container images, client bundles	Read access to storage, images or client downloads	27001 A.8.3, A.8.24; AIC-04, AIC-10

VULNERABLE PATTERN · SPOT IT IN REVIEW, NEVER SHIP IT

VULNERABLE · PYTHON 3.12vuln/py/t07.py

```
# VULNERABLE (T07): weights readable by anyone
import boto3

s3 = boto3.client("s3")
s3.upload_file("model.safetensors", "acme-models",
               "prod/model.safetensors",
               ExtraArgs={"ACL": "public-read"})
# The inference role also holds s3:* on every bucket,
# and the weights are not encrypted with a KMS key.
```

VULNERABLE · TYPESCRIPT 5 · NODE 22vuln/ts/t07.ts

```
// VULNERABLE (T07): the model ships to every browser
export async function loadPricingModel() {
  // proprietary weights served as a static file
  return ort.InferenceSession.create(
    "/static/models/pricing-v4.onnx");
}
// Anyone can save it from the browser's network tab.
// Keep valuable models server-side behind an API.
```

Code review checklist

- 01 No public ACLs or policies on model buckets
- 02 Inference roles read only the model they serve
- 03 Weights are encrypted with a KMS key with its own policy
- 04 Valuable models are not shipped to browsers or devices

Tests that prove the fix

- 01 A policy check fails on any public grant
- 02 The inference role cannot read other model prefixes
- 03 Client bundles contain no model files

THE FIX Keep weights private and encrypted, scope each role to one model, and serve valuable models server-side.

T08

AI SECURITY

THREAT

Direct training data leak

Training copies of production data land in scratch buckets, notebooks, CI runners or laptops with weaker controls than the source system.

WHERE IT SHOWS UP	ATTACKER NEEDS	FIX WITH
Data export jobs, notebooks, feature stores, CI artefacts	Access to wherever the training copy lands	PC-03, AIC-01, AIC-09

VULNERABLE PATTERN · SPOT IT IN REVIEW, NEVER SHIP IT

VULNERABLE · PYTHON 3.12vuln/py/t08.py

```
# VULNERABLE (T08): production PII copied for training
import pandas as pd

df = pd.read_sql("SELECT * FROM customers", prod_conn)
df.to_parquet("s3://ml-scratch/customers.parquet")
# Every column of every customer, unclassified, no
# KMS key, readable by the whole data team.
```

VULNERABLE · TYPESCRIPT 5 · NODE 22vuln/ts/t08.ts

```
// VULNERABLE (T08): production PII dumped for training
import { writeFile } from "node:fs/promises";

export async function exportTrainingSet(db: Db) {
  const rows = await db.query("SELECT * FROM customers");
  await writeFile("./data/train.jsonl", rows
    .map((r: unknown) => JSON.stringify(r)).join("\n"));
}
// Every column of every customer on a laptop or CI
// runner, unclassified and unencrypted.
```

Code review checklist

- 01 No SELECT * exports from production for training
- 02 Training data keeps the source's classification and encryption
- 03 Only DPIA-approved features are extracted
- 04 Exports are logged and expire

Tests that prove the fix

- 01 The export fails on a column not on the allow-list
- 02 The training bucket rejects unencrypted writes
- 03 Scratch storage deletes data after the set period

THE FIX Extract only approved features, pseudonymise with a key, and keep training data under the source system's controls.

**T09**

AI SECURITY

THREAT

Model input leak

Prompts carry whole records, documents or secrets to an outside model and into logs, often on consumer terms that allow retention or training.

WHERE IT SHOWS UP

LLM client wrappers, logging
middleware, copilots, browser
extensions

ATTACKER NEEDS

None needed: the leak is built in

FIX WITH

PC-06, PC-03

VULNERABLE PATTERN · SPOT IT IN REVIEW, NEVER SHIP IT

VULNERABLE · PYTHON 3.12

vuln/py/t09.py

```
# VULNERABLE (T09): whole record sent out and logged
import json
import logging

def summarise(customer: dict, llm) -> str:
    prompt = ("Summarise this customer:\n"
              + json.dumps(customer)) # DOB, IBAN, notes
    logging.info("prompt=%s", prompt) # now in logs too
    return llm.complete(prompt) # consumer-tier terms
```

VULNERABLE · TYPESCRIPT 5 · NODE 22

vuln/ts/t09.ts

```
// VULNERABLE (T09): whole record sent out and logged
export async function summarise(customer: Customer) {
    const prompt = "Summarise this customer:\n" +
        JSON.stringify(customer); // DOB, IBAN, notes
    console.log("prompt", prompt); // now in logs too
    return llm.complete(prompt); // consumer-tier terms
}
```

Code review checklist

- 01 Only the fields the task needs go into the prompt
- 02 Prompts are redacted before logging
- 03 The provider is approved, on enterprise terms
- 04 Region and retention match the DPIA

Tests that prove the fix

- 01 The prompt builder emits only allow-listed fields
- 02 Log records contain no emails, cards or phone numbers
- 03 The client refuses an unapproved region

THE FIX Minimise prompt content, redact before logging, and use an approved provider with set retention and region.



T10

AI SECURITY

THREAT

Sensitive model output

Retrieval returns documents the user may not see, or the prompt holds a secret the model can be talked into repeating.

WHERE IT SHOWS UP

RAG retrieval, system prompts, tool output fed back to the model

ATTACKER NEEDS

An ordinary user asking the right question

FIX WITH

AIC-12, AIC-06

VULNERABLE PATTERN · SPOT IT IN REVIEW, NEVER SHIP IT

VULNERABLE · PYTHON 3.12

vuln/py/t10.py

```
# VULNERABLE (T10): retrieval ignores who is asking
import os

SYSTEM = ("You are the support bot. Billing key: "
          + os.environ["BILLING_KEY"]) # secret in prompt

def answer(question: str, index, llm) -> str:
    docs = index.search(question, k=5) # every tenant
    context = "\n".join(d.text for d in docs)
    return llm.chat(system=SYSTEM,
                    user=f"{context}\n\nQ: {question}")
```

VULNERABLE · TYPESCRIPT 5 · NODE 22

vuln/ts/t10.ts

```
// VULNERABLE (T10): retrieval ignores who is asking
const SYSTEM = "You are the support bot. Billing key: " +
  process.env.BILLING_KEY; // secret in the prompt

export async function answer(question: string) {
  const docs = await index.search(question, 5); // all
  const context = docs.map((d: Doc) => d.text)
    .join("\n");
  return llm.chat(SYSTEM, `${context}\n\nQ: ${question}`);
}
```

Code review checklist

- 01 Retrieval filters by the requester's rights before ranking
- 02 No secrets or credentials in prompts or context
- 03 Index entries carry document-level access metadata
- 04 Outputs are checked for secret patterns

Tests that prove the fix

- 01 A tenant B user gets no tenant A documents
- 02 A document holding a key pattern blocks the answer
- 03 Prompt templates pass a secret scan in CI

THE FIX Enforce permissions at retrieval and keep secrets out of prompts entirely.

T11

AI SECURITY

THREAT

Prompt injection

Untrusted text from web pages, emails, files or retrieved documents is mixed with instructions, and the model holds tools that act on the world.

WHERE IT SHOWS UP	ATTACKER NEEDS	FIX WITH
Agents, RAG pipelines, email and document assistants	Ability to place text where the model will read it	AIC-10, AIC-06, AIC-13, AIC-16

VULNERABLE PATTERN · SPOT IT IN REVIEW, NEVER SHIP IT

VULNERABLE · PYTHON 3.12vuln/py/t11.py

```
# VULNERABLE (T11): web text becomes instructions
def handle(task: str, url: str, agent) -> str:
    page = fetch_text(url) # attacker-owned
    return agent.run(
        instructions=f"{task}\n\n{page}", # mixed in
        tools=[send_email, refund_order, run_sql],
        auto_approve=True) # no human step
```

VULNERABLE · TYPESCRIPT 5 · NODE 22vuln/ts/t11.ts

```
// VULNERABLE (T11): web text becomes instructions
export async function handle(task: string, url: string) {
    const page = await (await fetch(url)).text(); // hostile
    return agent.run({
        instructions: `${task}\n\n${page}`, // mixed together
        tools: [sendEmail, refundOrder, runSql],
        autoApprove: true, // no human step
    });
}
```

Code review checklist

- 01 Untrusted content goes in as data, never merged into instructions
- 02 Each agent has its own identity and minimum tools
- 03 Irreversible actions need human approval from the UI
- 04 Retrieved content is screened and quarantined on findings

Tests that prove the fix

- 01 An injection corpus with a planted canary leaks nothing
- 02 An agent cannot call a tool outside its grant
- 03 An irreversible tool without approval is refused

THE FIX Least privilege per agent, human approval for irreversible actions, screening of untrusted content and red teaming.



T12

AI SECURITY

THREAT

Output contains injection attacks

Model output is trusted and passed to a browser, database, shell or another service, so injected content runs there.

WHERE IT SHOWS UP

Chat UIs, markdown renderers, text-to-SQL, code and command generation

ATTACKER NEEDS

Anyone who can steer the output, often through prompt injection

FIX WITH

AIC-11, AIC-10

VULNERABLE PATTERN · SPOT IT IN REVIEW, NEVER SHIP IT

VULNERABLE · PYTHON 3.12

vuln/py/t12.py

```
# VULNERABLE (T12): model output executed downstream
import subprocess

def report(question: str, llm, db) -> list:
    sql = llm.complete(f"Write SQL for: {question}")
    rows = db.execute(sql).fetchall() # runs anything
    cmd = llm.complete("Shell command to zip the report")
    subprocess.run(cmd, shell=True) # runs anything
    return rows
```

VULNERABLE · TYPESCRIPT 5 · NODE 22

vuln/ts/t12.ts

```
// VULNERABLE (T12): model output rendered as HTML
export function showAnswer(el: HTMLElement,
    reply: string) {
    el.innerHTML = marked.parse(reply);
}
// Scripts, event handlers, images and links to any
// host all render. A markdown image can carry chat
// data to an attacker's server in its URL.
```

Code review checklist

- 01 Output never reaches innerHTML, eval, a shell or raw SQL
- 02 Rendered markdown drops images and off-list links
- 03 Generated SQL maps to parameterised templates
- 04 A content security policy blocks inline scripts

Tests that prove the fix

- 01 Script and event-handler payloads render as text
- 02 A markdown image to an outside host is removed
- 03 A generated DROP statement is rejected

THE FIX Escape and validate output for each consumer, allow links to approved hosts only, and never execute model text.

**P01**

PRIVACY

THREAT

Property and attribute inference

Features that proxy protected traits go into a model, and its outputs let anyone infer those traits about a person.

WHERE IT SHOWS UP

Feature engineering, scoring APIs, analytics models

ATTACKER NEEDS

Query access plus some background facts

FIX WITH

PC-03, AIC-07, AIC-09

VULNERABLE PATTERN · SPOT IT IN REVIEW, NEVER SHIP IT

VULNERABLE · PYTHON 3.12

vuln/py/p01.py

```
# VULNERABLE (P01): sensitive proxies in, raw score out
FEATURES = ["age", "postcode", "first_name",
            "pharmacy_spend", "loan_amount"]
# first_name and postcode can proxy ethnicity and
# pharmacy_spend can proxy health. Nobody tested what
# the model reveals about them.

def score(model, applicant: dict) -> dict:
    x = [applicant[f] for f in FEATURES]
    return {"risk": model.predict_proba(x)[0][1]}
```

VULNERABLE · TYPESCRIPT 5 · NODE 22

vuln/ts/p01.ts

```
// VULNERABLE (P01): sensitive proxies in, raw score out
const FEATURES = ["age", "postcode", "firstName",
                  "pharmacySpend", "loanAmount"];
// firstName and postcode can proxy ethnicity and
// pharmacySpend can proxy health. Nobody tested
// what the model reveals about them.

export function score(model: Model, a: Applicant) {
    const x = FEATURES.map((f) => a[f]);
    return { risk: model.predictProba(x)[0][1] };
}
```

Code review checklist

- 01 Every feature is on the DPIA allow-list
- 02 Proxies such as postcode, first name and spend categories are justified or removed
- 03 Outputs are labels or bands
- 04 An attribute inference test runs before release

Tests that prove the fix

- 01 The pipeline fails on a special-category or unlisted feature
- 02 Outputs cannot predict the protected trait above the set level
- 03 The API returns no raw scores

THE FIX Minimise features against the DPIA list, return coarse outputs and test for attribute inference.

P02

PRIVACY

THREAT

Re-identification of de-identified data

Data is called anonymous after names are dropped and emails hashed, while date of birth, postcode and other quasi-identifiers stay.

WHERE IT SHOWS UP	ATTACKER NEEDS	FIX WITH
Data sharing, analytics exports, training and test sets	Anyone with another dataset to link against	PC-02, PC-03

VULNERABLE PATTERN · SPOT IT IN REVIEW, NEVER SHIP IT

VULNERABLE · PYTHON 3.12vuln/py/p02.py

```
# VULNERABLE (P02): "anonymised" but re-identifiable
import hashlib

def anonymise(row: dict) -> dict:
    row = dict(row)
    row.pop("name")
    row["email"] = hashlib.sha256(                # reversed by
        row["email"].encode()).hexdigest()      # guessing
    return row # keeps date_of_birth, postcode and sex
# Then shared with a partner as "anonymous data".
```

VULNERABLE · TYPESCRIPT 5 · NODE 22vuln/ts/p02.ts

```
// VULNERABLE (P02): "anonymised" but re-identifiable
import { createHash } from "node:crypto";

export function anonymise(row: CustomerRow) {
    const { name, ...rest } = row;
    return { ...rest, email: createHash("sha256")
        .update(row.email).digest("hex") }; // guessable
}
// date_of_birth, postcode and sex are kept, then the
// file goes to a partner as "anonymous data".
```

Code review checklist

- 01 Plain hashes of identifiers are not treated as anonymisation
- 02 Quasi-identifiers are generalised before release
- 03 k-anonymity and uniqueness are measured
- 04 A motivated intruder test is on file

Tests that prove the fix

- 01 Release fails when k drops below the agreed minimum
- 02 The unique-row share is reported
- 03 Hashed identifiers are flagged by the schema check

THE FIX Measure k and uniqueness, generalise or suppress, use keyed pseudonyms, and treat the result as personal data unless testing shows otherwise.



PRIVACY

THREAT

P03 Unlawful purpose or legal basis for training

Code reuses whatever data is available for a new AI purpose without checking purpose, lawful basis, consent or opt-outs.

WHERE IT SHOWS UP

Fine-tuning dataset builders, analytics jobs, data lake queries

ATTACKER NEEDS

None: a compliance failure

FIX WITH

PC-01, PC-04

VULNERABLE PATTERN • SPOT IT IN REVIEW, NEVER SHIP IT

VULNERABLE • PYTHON 3.12

vuln/py/p03.py

```
# VULNERABLE (P03): reused because the data is there
def build_finetune_set(db) -> list[dict]:
    # Collected to resolve support cases, now used to
    # train a sales model. No DPIA, no basis check, no
    # notice, and opt-outs are ignored.
    return db.all("SELECT body, customer_id "
                  "FROM support_tickets")
```

VULNERABLE • TYPESCRIPT 5 • NODE 22

vuln/ts/p03.ts

```
// VULNERABLE (P03): reused because the data is there
export async function buildFinetuneSet(db: Db) {
    // Collected to resolve support cases, now used to
    // train a sales model. No DPIA, no basis check, no
    // notice, and opt-outs are ignored.
    return db.all(
        "SELECT body, customer_id FROM support_tickets");
}
```

Code review checklist

- 01 The job declares a purpose matching an approved DPIA
- 02 The lawful basis is recorded; basis (f) has an LIA
- 03 Opt-out and consent flags are applied in the query
- 04 Special-category data has an Article 9 condition

Tests that prove the fix

- 01 A job with an unmatched purpose refuses to start
- 02 Basis (f) without an LIA fails
- 03 Opted-out records never reach the training set

THE FIX Gate training jobs on a DPIA record with purpose, basis and LIA, and filter opt-outs in the query.

LAW AND REGULATION

GDPR Arts 5(1)(b), 6, 6(4), 13, 14, 21

STANDARDS

ISO/IEC 42001 A.5.2, A.7.3; ISO/IEC 27701:2025

RELATED 27090 THREATS

None (compliance risk)

P04

PRIVACY

THREAT

Erasure and rectification failure

Erasure and correction requests reach the main database but not the vector index, logs, training files or the model's outputs.

WHERE IT SHOWS UP	ATTACKER NEEDS	FIX WITH
DSAR handlers, vector stores, log pipelines, training manifests	None: a rights failure	PC-04

VULNERABLE PATTERN · SPOT IT IN REVIEW, NEVER SHIP IT

VULNERABLE · PYTHON 3.12vuln/py/p04.py

```
# VULNERABLE (P04): erasure stops at the main database
def erase_user(user_id: str, db) -> None:
    db.execute("DELETE FROM users WHERE id = ?",
               (user_id,))
    # Still present: vector index chunks, prompt logs,
    # fine-tuning files, eval sets and model outputs
    # that name the person. Nothing suppresses them.
```

VULNERABLE · TYPESCRIPT 5 · NODE 22vuln/ts/p04.ts

```
// VULNERABLE (P04): erasure stops at the main database
export async function eraseUser(userId: string, db: Db) {
    await db.run("DELETE FROM users WHERE id = ?",
                 [userId]);
    // Still present: vector index chunks, prompt logs,
    // fine-tuning files, eval sets and model outputs
    // that name the person. Nothing suppresses them.
}
```

Code review checklist

- 01 Every store holding personal data implements erase()
- 02 The erasure job reports counts per store and fails loudly
- 03 A suppression list filters outputs about erased people
- 04 Retraining or unlearning timelines are written down

Tests that prove the fix

- 01 A seeded person is removed from every store
- 02 One failing store fails the job
- 03 Suppressed names never appear in outputs

THE FIX Orchestrate erasure across every AI store, suppress at output at once, and retrain on schedule.

**P05**

PRIVACY

THREAT

Unlawful solely automated decisions

A model score triggers a significant decision, such as a decline, with no meaningful human review, no reasons and no route to contest.

WHERE IT SHOWS UP

Credit, hiring, insurance, fraud and eligibility workflows

ATTACKER NEEDS

None: a design failure

FIX WITH

PC-05, PC-01

VULNERABLE PATTERN · SPOT IT IN REVIEW, NEVER SHIP IT

VULNERABLE · PYTHON 3.12

vuln/py/p05.py

```
# VULNERABLE (P05): the model decides, nobody reviews
def process_application(app: dict, model, mailer):
    score = model.predict_proba([features(app)])[0][1]
    if score < 0.3:
        decline(app["id"])
        mailer.send(app["email"], "Application declined")
    # no reasons, no human review, no way to contest
```

VULNERABLE · TYPESCRIPT 5 · NODE 22

vuln/ts/p05.ts

```
// VULNERABLE (P05): the model decides, nobody reviews
export async function processApplication(app: App) {
    const score = model.predictProba([features(app)])[0][1];
    if (score < 0.3) {
        await decline(app.id);
        await mailer.send(app.email, "Application declined");
    }
    // no reasons, no human review, no way to contest
}
```

Code review checklist

- 01 Adverse or borderline outcomes go to a person who can change them
- 02 Reasons are recorded and explainable
- 03 The notice says a system was involved and how to contest
- 04 Override rates are monitored

Tests that prove the fix

- 01 A low score creates a review task, never an automatic decline
- 02 Every adverse outcome carries reasons and a contest route
- 03 The override-rate metric is emitted

THE FIX Route adverse and borderline cases to human review, send notice and reasons, and monitor overrides.

LAW AND REGULATION

GDPR Art 22; UK GDPR Arts 22A to 22D; AI Act Art 14

STANDARDS

ISO/IEC 42001 A.9.2, A.5.4

RELATED 27090 THREATS

None (compliance risk)

PRIVACY

THREAT

P06

Excessive retention and uncontrolled transfer

Prompts and replies are logged in full, kept forever and processed in whatever region the SDK defaults to.

WHERE IT SHOWS UP	ATTACKER NEEDS	FIX WITH
LLM client configuration, logging, analytics and support tools	None: a compliance failure that widens any breach	PC-06

VULNERABLE PATTERN · SPOT IT IN REVIEW, NEVER SHIP IT

VULNERABLE · PYTHON 3.12vuln/py/p06.py

```
# VULNERABLE (P06): everything kept, anywhere, forever
import logging

client = LLMClient(region="us-east-1") # the default

def chat(prompt: str) -> str:
    reply = client.complete(prompt)
    logging.info({"prompt": prompt, "reply": reply})
    # Log store has no TTL, provider retention unknown,
    # no transfer assessment for this region.
    return reply
```

VULNERABLE · TYPESCRIPT 5 · NODE 22vuln/ts/p06.ts

```
// VULNERABLE (P06): everything kept, anywhere, forever
const client = new LlmClient({ region: "us-east-1" });

export async function chat(prompt: string) {
    const reply = await client.complete(prompt);
    console.log(JSON.stringify({ prompt, reply }));
    // Log store has no TTL, provider retention unknown,
    // no transfer assessment for this region.
    return reply;
}
```

Code review checklist

- 01 The region is set explicitly and approved
- 02 Prompts and replies are redacted before logging
- 03 Every log record has an expiry
- 04 Provider retention and training terms are on file

Tests that prove the fix

- 01 The client refuses an unapproved region
- 02 Redaction removes emails, card and phone numbers
- 03 Records older than the TTL are gone

THE FIX Pin regions, redact before logging, set TTLs and use provider terms with limited retention.

**R01**

RESILIENCE

THREAT

Energy-latency (sponge) inputs

Requests have no limits on input size, output tokens or time, so crafted inputs consume maximum compute and slow the service for everyone.

WHERE IT SHOWS UP

Inference endpoints and LLM gateways

ATTACKER NEEDS

Ability to send requests

FIX WITH

RC-01, AIC-08

VULNERABLE PATTERN · SPOT IT IN REVIEW, NEVER SHIP IT

VULNERABLE · PYTHON 3.12

vuln/py/r01.py

```
# VULNERABLE (R01): no limits on size, output or time
def answer(user_text: str, llm) -> str:
    # 2 MB of pasted input, no max_tokens, no timeout:
    # one crafted request can hold a GPU for minutes
    return llm.generate(prompt=user_text)
```

VULNERABLE · TYPESCRIPT 5 · NODE 22

vuln/ts/r01.ts

```
// VULNERABLE (R01): no limits on size, output or time
export async function answer(userText: string) {
    // 2 MB of pasted input, no maxTokens, no timeout:
    // one crafted request can hold a GPU for minutes
    return llm.generate({ prompt: userText });
}
```

Code review checklist

- 01 Input size is capped before tokenising
- 02 max_tokens is always set and clamped
- 03 Every model call has a timeout
- 04 Rate limits apply per identity

Tests that prove the fix

- 01 Oversized input is rejected
- 02 A max_tokens request above the cap is clamped
- 03 A slow model call times out

THE FIX Cap input and output, set timeouts and rate-limit per identity.

**R02**

RESILIENCE

THREAT

Cost exhaustion and LLMjacking

A paid model key leaks or calls go unmetered, so attackers or bugs run up costs until budgets or quotas stop the service.

WHERE IT SHOWS UP

Client bundles, CI secrets, batch jobs, multi-tenant APIs

ATTACKER NEEDS

A leaked key, or any user of an unmetered feature

FIX WITH

RC-02, AIC-08

VULNERABLE PATTERN · SPOT IT IN REVIEW, NEVER SHIP IT

VULNERABLE · PYTHON 3.12

vuln/py/r02.py

```
# VULNERABLE (R02): long-lived key, no cap, no alerts
LLM_KEY = "<long-lived key committed to the repo>"

def batch_summarise(docs: list[str], llm) -> list[str]:
    # unbounded loop over user-supplied documents on a
    # premium model, with no per-tenant budget or quota
    return [llm.complete(d, key=LLM_KEY,
                        model="premium-large")
            for d in docs]
```

VULNERABLE · TYPESCRIPT 5 · NODE 22

vuln/ts/r02.ts

```
// VULNERABLE (R02): a paid key shipped to browsers
const client = new LlmClient({
  apiKey: process.env.NEXT_PUBLIC_LLM_API_KEY,
});
// Next.js inlines NEXT_PUBLIC_ variables into the
// browser bundle, so anyone can lift the key. No
// per-user budget, no spend alert, no hard cap.
export const ask = (q: string) => client.complete(q);
```

Code review checklist

- 01 No model keys in client code or NEXT_PUBLIC_ variables
- 02 Keys are scoped, vaulted and rotated
- 03 Spend is reserved per tenant before each call
- 04 The provider account has budget alerts and hard caps

Tests that prove the fix

- 01 The build fails if a key pattern appears in the client bundle
- 02 A tenant over its cap is refused
- 03 An alert fires at the set share of budget

THE FIX Keep keys server-side, reserve spend per tenant before each call, and set provider budgets with hard caps.

**R03**

RESILIENCE

THREAT

Availability poisoning

Automatic retraining on unvetted data ships a degraded model with no gate and no way back.

WHERE IT SHOWS UP

Scheduled retraining and continuous learning pipelines

ATTACKER NEEDS

Ability to influence training data, such as feedback

FIX WITH

AIC-02, RC-05, AIC-01

VULNERABLE PATTERN · SPOT IT IN REVIEW, NEVER SHIP IT

VULNERABLE · PYTHON 3.12

vuln/py/r03.py

```
# VULNERABLE (R03): auto-retrain and auto-ship nightly
def nightly(job) -> None:
    data = job.load("feedback/yesterday/*") # unvetted
    model = job.train(data)
    job.registry.set_production(model) # no gate
    job.registry.delete_old_versions() # no rollback
```

VULNERABLE · TYPESCRIPT 5 · NODE 22

vuln/ts/r03.ts

```
// VULNERABLE (R03): auto-retrain and auto-ship nightly
export async function nightly(job: Job) {
    const data = await job.load("feedback/yesterday/*");
    const model = await job.train(data); // unvetted data
    await job.registry.setProduction(model); // no gate
    await job.registry.deleteOldVersions(); // no rollback
}
```

Code review checklist

- 01 Retraining data passes provenance checks
- 02 Promotion needs the golden-set gate
- 03 The last known good model is kept deployable
- 04 Old versions are never deleted automatically

Tests that prove the fix

- 01 A model trained on a seeded degraded set fails the gate
- 02 Rollback restores the previous version
- 03 The registry keeps at least one prior version

THE FIX Gate every promotion, keep the last known good model and roll back on quality drops.

R04

RESILIENCE

THREAT

Silent drift and degradation

A model is deployed once and never measured again, so its decisions degrade as the data changes.

WHERE IT SHOWS UP

Any long-lived production model

ATTACKER NEEDS

None: the world changes

FIX WITH

AIC-15, RC-05

VULNERABLE PATTERN · SPOT IT IN REVIEW, NEVER SHIP IT

VULNERABLE · PYTHON 3.12

vuln/py/r04.py

```
# VULNERABLE (R04): trained once, trusted forever
MODEL = load_model("credit-2023-03.onnx")

def decide(applicant: dict) -> str:
    score = MODEL.score(applicant)
    return "approve" if score > 0.6 else "refer"
# No input drift metric, no outcome sampling and no
# re-validation date: quality decays silently.
```

VULNERABLE · TYPESCRIPT 5 · NODE 22

vuln/ts/r04.ts

```
// VULNERABLE (R04): trained once, trusted forever
const MODEL = loadModel("credit-2023-03.onnx");

export function decide(applicant: Applicant) {
    return MODEL.score(applicant) > 0.6
        ? "approve" : "refer";
}
// No input drift metric, no outcome sampling and no
// re-validation date: quality decays silently.
```

Code review checklist

- 01 Input or score drift is measured against the release baseline
- 02 Ground truth is sampled from live traffic
- 03 A re-validation date is set
- 04 Rollback is wired to quality thresholds

Tests that prove the fix

- 01 PSI above the limit triggers re-validation
- 02 Quality below the floor triggers rollback
- 03 A model past its re-validation date fails the scheduled check

THE FIX Monitor drift and outcome quality, re-validate on triggers and on schedule, and roll back automatically.

R05

RESILIENCE

THREAT

Third-party model outage or silent change

Code calls a floating model alias at one provider with no timeout or fallback, so outages stop the process and silent updates change behaviour.

WHERE IT SHOWS UP	ATTACKER NEEDS	FIX WITH
LLM API clients and provider configuration	None: dependency risk	RC-04, RC-03

VULNERABLE PATTERN · SPOT IT IN REVIEW, NEVER SHIP IT

VULNERABLE · PYTHON 3.12vuln/py/r05.py

```
# VULNERABLE (R05): floating alias, one provider
def classify(ticket: str, client) -> str:
    reply = client.complete(
        model="vendor-large-latest", # moves under you
        prompt=ticket)              # no timeout
    return reply.text # no regression suite, no plan B
```

VULNERABLE · TYPESCRIPT 5 · NODE 22vuln/ts/r05.ts

```
// VULNERABLE (R05): floating alias, one provider
export async function classify(ticket: string) {
    const reply = await client.complete({
        model: "vendor-large-latest", // moves under you
        prompt: ticket,               // no timeout
    });
    return reply.text; // no regression suite, no plan B
}
```

Code review checklist

- 01 Model IDs are pinned to a dated or versioned release
- 02 Every call has a timeout
- 03 A fallback path exists and is tested
- 04 A regression suite runs before any model change

Tests that prove the fix

- 01 Floating aliases fail configuration validation
- 02 A primary timeout routes to the fallback
- 03 The suite blocks a change that lowers the pass rate

THE FIX Pin versions, regression-test changes, set timeouts and fail over to a tested fallback.

R06

RESILIENCE

THREAT

Cascading failure in agent chains

Agent loops have no step, token or retry limits, so one bad step repeats, spawns more agents and spreads the damage.

WHERE IT SHOWS UP	ATTACKER NEEDS	FIX WITH
Agent frameworks, multi-agent orchestration, tool loops	Not needed, though injection can trigger it	RC-06, AIC-16, AIC-10

VULNERABLE PATTERN · SPOT IT IN REVIEW, NEVER SHIP IT

VULNERABLE · PYTHON 3.12vuln/py/r06.py

```
# VULNERABLE (R06): an unbounded agent loop
def run(agent, goal: str) -> str:
    state = agent.start(goal)
    while not state.done:
        try:
            state = agent.step(state) # no step cap
        except Exception:
            continue # may spawn agents
    return state.answer # retries forever
```

VULNERABLE · TYPESCRIPT 5 · NODE 22vuln/ts/r06.ts

```
// VULNERABLE (R06): an unbounded agent loop
export async function run(goal: string) {
    let state = await agent.start(goal);
    while (!state.done) {
        try {
            state = await agent.step(state); // no step cap
        } catch {
            continue; // spawns agents
        }
    }
    return state.answer; // retries forever
}
```

Code review checklist

- 01 Every run has step, token and repeat limits
- 02 Retries are capped with back-off
- 03 Outputs are validated before the next agent uses them
- 04 A kill switch can stop the workflow

Tests that prove the fix

- 01 A looping task stops at the step limit
- 02 Repeated identical tool calls are blocked
- 03 The kill switch stops tools mid-run

THE FIX Bound every run, validate between stages and wire in a kill switch.