ENGINEERING FOR EUROPE'S AI REGULATION

EU AI Act for Developers

Compliance engineering in Python: from legal requirement to audit-ready evidence

Robert Barcik

First edition, July 2026

EU AI Act for Developers · First edition, July 2026

This book is the companion textbook to the online course “EU AI Act for Developers: Compliance Engineering in Python”.

This book is engineering guidance, not legal advice. The EU AI Act and its guidelines continue to evolve; for decisions with legal exposure, consult your legal representative.

Introduction

Before anything else: this book is engineering guidance, not legal advice. I am an engineer and an educator. I am not your lawyer, and reading this book does not create one. Everything here rests on careful research, official texts, and published Commission guidance, turned into practices you can run. Where the law is settled, I say so. Where it is still moving, I say that too, and I flag the difference every time it matters. For any decision you would stake a product launch, a hiring pipeline, or a regulatory filing on, bring in counsel. What this book promises instead is narrower and, for an engineer, more useful: when your counsel or your auditor asks for evidence, you will already have it.

What this book is

This is the companion text to the video course “EU AI Act for Developers: Compliance Engineering in Python.” The course teaches the Act by building. This book covers the same ground in more depth, with the full argument written out, the tables expanded, and room for the caveats a spoken lecture has to compress.

The premise is simple: the EU AI Act, read by a builder rather than a lawyer, is a requirements document. It describes a system’s inputs, its documentation, its logging, its human-oversight behavior, and its acceptance criteria. It just writes all of that in legal prose instead of user stories. This book does the translation once, chapter by chapter, obligation by obligation, so that later chapters can go straight to code.

Who it is for

You build or integrate AI systems and at least one of them touches the EU market, now or eventually. You do not need a legal background. You need working Python and the willingness to read a regulation the way you read a long, oddly worded specification document: closely, once, with a highlighter. If you have taken the companion introduction course on the AI Act, you already have deeper legal context, and it will

help. It is not required. This book's first chapter carries the legal minimum a developer needs.

What this book deliberately does not do: teach conformity-assessment law in depth, coach you through organizational change management, or promise legal certainty the law itself does not yet have. Those are real needs. They are met elsewhere, including in the introduction course this one follows.

How it pairs with the videos

Each chapter mirrors one module of the course. The slides and the spoken narration give you the shape of an obligation and the shortest path to a working artifact. This book gives you the full argument: the statutory language, the guidance documents behind it, the worked examples, and the honest edge cases the video compresses for time. Watch the module, then read the chapter, or read first and watch second. Either order works. The two are not duplicates of each other; the video is the fast pass, the book is the reference you come back to when you are actually writing the classification memo or the logging schema at your desk.

How to read it

Read Chapters 1 and 2 in order. They are the legal minimum: what the Act asks of you, and whether it asks anything of your system at all. From Chapter 3 onward, each chapter stands on its own, built around one article or obligation, so you can jump to whichever one just landed on your desk. Three fictional systems, introduced in the classification chapter, carry through every later chapter as worked examples: a customer-support chatbot, a CV-screening tool, and a predictive-maintenance model. They cover the three answers a real classification exercise produces, and each chapter's exercises assume you are also working a system of your own alongside them.

Every chapter ends with a worksheet. Fill it in for your own system, not just the fictional ones. The worksheets, taken together across the book, are the first draft of your evidence pack: the folder of artifacts an auditor, a client, or your own future self will eventually ask for.

Along the way you will meet four kinds of boxes, and they encode a distinction engineers should care about. **The Act says** boxes hold the statutory language itself; treat their contents like an API contract. **My take** boxes are my judgment where the law leaves room; treat them like a senior colleague's code review comment. **Watch out** boxes flag the mistakes teams actually make. **As of** boxes carry facts with an expiry date, such as enforcement status or pending guidance; re-verify those before you build a decision on them. Everything outside a box is explanation; the boxes tell you what kind of claim you are reading.

The promise, restated

If a topic in this book cannot end in something you ran, it does not belong in this book. That is the filter. Chapter 1 starts with the map.

CHAPTER 1

From Legal Requirement to Technical Specification

Lawyers have AI Act courses written for them. Managers have AI Act courses written for them. When a developer asks the only question that matters to a developer, “so what do I actually build?”, the usual answer is a bullet list and a wish of good luck. This chapter, and the course it introduces, exist to close that gap.

The claim underneath everything that follows: the EU AI Act, read the way an engineer reads a specification instead of the way a lawyer reads a statute, is a requirements document. It has functional requirements (log these events, test for these failure modes). It has non-functional requirements (accuracy, robustness, security levels described in outcome terms). It has documentation requirements (Annex IV, section by section). It even has acceptance criteria, buried in words like “appropriate” and “proportionate” that need translating into numbers before they mean anything to a test suite. This chapter does that translation once, at the level of the whole Act, so every later chapter can go straight to building.

The disclaimer, up front

Two things matter before anything else. First, the promise: every obligation this course covers ends in something you run on your own machine. Logging becomes a schema you implement. Accuracy becomes an eval pipeline that executes. Documentation becomes a generated skeleton, not a blank template. If a topic cannot become an artifact, it does not belong in this course; the introduction course covers it instead.

MY TAKE

Second, the disclaimer, stated as plainly as I can state it: I am an engineer and an educator, not your lawyer. This book turns careful research into engineering practice, and it will always tell you where the law is settled and where it is still moving. For decisions with real legal exposure, bring in counsel. What I can promise is that when your counsel or your auditor asks for evidence, you will be the person in the room who already has it. Take everything below with that grain of salt.

The Act in one map, for builders

The enacted Act has thirteen chapters. As a builder, you live in three of them, and it helps to know why the other ten exist even if you never open them.

CHAPTER	SUBJECT	WHAT A BUILDER NEEDS FROM IT
I	General provisions: scope, definitions	The definitions decide whether anything else in the Act applies to you. Start here every time.
II	Prohibited practices	A short list of banned techniques (manipulation, social scoring, some biometric uses). For most engineering teams this is a checklist to verify against, not a system to design around.
III	High-risk AI systems	Your home chapter. Classification (Article 6, Annex III) and the requirements that become code: Articles 8 through 15, plus the provider and deployer obligations in Articles 16 through 27.
IV	Transparency obligations	Chatbot disclosure, synthetic-content marking. Small builds, some of the nearest deadlines.
V	General-purpose AI models	Binds people who build or substantially modify a model. Calling one through an API does not put you here.

CHAPTER	SUBJECT	WHAT A BUILDER NEEDS FROM IT
VI	Measures in support of innovation	Regulatory sandboxes, support for SMEs. Know it exists.
VII	Governance	The AI Office, the Board, the scientific panel, national authorities. Machinery above your system.
VIII	EU database	Where high-risk systems get registered. One form you will eventually fill in.
IX	Post-market monitoring, information sharing, market surveillance	Where your logs and incident reports go once the system ships. Covered in this course's later modules.
X	Codes of conduct	Voluntary frameworks for systems that are not high-risk.
XI	Delegation of power and committee procedure	Legislative plumbing.
XII	Penalties	Up to seven percent of global turnover or thirty-five million euro, whichever is higher, for the worst violations. In force since August 2025.
XIII	Final provisions	Entry into force, amendments, review clauses.

One terminology note that will save you confusion when you read older material: draft versions of the Act organized these into "Titles." The enacted text uses "Chapters." If a source says "Title III," it is citing pre-final numbering; the content usually still maps to what is now Chapter III. This book and course use enacted terminology throughout: Chapters, and "deployer" rather than the draft-era "user."

Chapter III is the one this course spends the most time in, because Articles 8 through 15 are the requirements that most directly become code: data governance, technical documentation, record-keeping, transparency to deployers, human oversight, accuracy, and robustness including cybersecurity. Chapter IV's transparency duties are lighter but land sooner. Chapter V matters mostly by exclusion: it tells you when you are *not* the one it is talking to, which is most of the time.

Provider and deployer: the roles the Act actually cares about

The Act does not distribute obligations by job title or company size. It distributes them by role, and the role is a property of a specific system in a specific use case, not a property of you or your employer.

Provider. You develop an AI system, or a general-purpose AI model, and place it on the market or put it into service under your own name or trademark. This is where the heavy obligations land: the whole of Articles 8 through 15 for high-risk systems, conformity assessment, technical documentation, registration in the EU database. Article 25 extends provider obligations to anyone who rebrands a system under their own name, makes a substantial modification to a high-risk system that keeps it high-risk, or changes an AI system's intended purpose in a way that makes it high-risk when it was not before. That last clause matters more than it looks: providership can attach to you even if you never wrote the original model.

Deployer. You use an AI system under your own authority in a professional context. The duties are real but lighter: use the system according to the provider's instructions, assign competent human oversight, keep the logs the system generates available to you, run a Fundamental Rights Impact Assessment for certain high-risk uses (Article 27), and tell the provider if something goes wrong. Deployers do not build the system's technical documentation. They do have to read it and act on it.

The catch, and the single most useful thing to internalize in this chapter: these are per-system, per-use-case assignments, not identities. The same company, sometimes the same team, is the provider of one system and the deployer of another. And the boundary moves with what you build next. A team that builds a screening tool for internal use is, for that system, arguably a deployer of a tool they also happen to have written. The moment that same team packages the tool and sells it to a client, they have become its provider, with the full weight of Articles 8 through 15 now pointing at them. Nothing about the code changed. The commercial relationship did, and the Act cares about the relationship, not the code.

A second, subtler version of the same trap: a system built and used purely in-house can still cross into "placing on the market" territory if it is made available to a different legal

entity, including a subsidiary, a client, or a partner organization, even without an exchange of money. “Placing on the market” and “putting into service” are defined broadly on purpose.

Chapter 2 of this course, on classification, takes three fictional systems (a customer-support chatbot, a CV-screening tool used inside applicant-tracking software, and a predictive-maintenance model for industrial pumps) through this exact question, in enough detail to be a template for your own systems. For now, the exercise at the end of this chapter asks you to make a first pass at the same question using only what you have read here.

The fine-tuning trap

This is the single most common question developers ask, so it gets answered in chapter one rather than buried later.

You are building on a frontier model. You call it through an API. You do retrieval-augmented generation over your own documents. You fine-tune it on your own data, the way most teams mean “fine-tune”: a modest supervised pass, a LoRA adapter, a few million tokens of domain data. Does any of that make you the provider of a general-purpose AI model, with model-level obligations like training-data summaries and systemic-risk assessments?

Almost always, no. API calls, prompting, retrieval augmentation, and ordinary fine-tuning leave the original provider as the model’s provider. What you are providing is an AI *system* built on top of someone else’s model, and that system has its own duties, which is what most of this course teaches. You do not inherit Chapter V’s model-level obligations by using a model well.

The Commission’s guidelines on GPAI obligations, published as C(2025) 7719 in November 2025, draw an indicative line for the harder case: a downstream actor who modifies or fine-tunes an existing general-purpose AI model becomes that model’s provider when the modification consumes more than roughly one third of the original model’s training compute. Think about what a third of a frontier training run actually

costs in compute-hours, and it becomes obvious why the Commission expects very few fine-tuners to cross it.

Two honesty notes belong here, and the course will repeat them every time this topic comes up:

1. **This is guidance, not statute.** The Act itself does not specify this threshold; the Commission's guidelines interpret Recital 97's acknowledgment that GPAI models "may be modified or fine-tuned into new models" without saying exactly where the line sits. Guidance can be updated, and it does not bind courts the way the Act's own text does.
2. **The one-third figure is a presumption, not the real test.** The guidelines are explicit that the underlying question is whether the modification substantially changed the model's generality, its capabilities, or the systemic risks it poses. Compute is a proxy for that question, a convenient one to estimate, but not the question itself. If your fine-tuning is small in compute terms but somehow unlocks a materially different capability profile, document your reasoning rather than leaning on the compute threshold alone.

When you are in the gray zone, the practical move is not to guess and hope. Write down your reasoning: what you changed, how much compute it took relative to a public estimate of the base model's training run, and why you concluded you did or did not cross into model-provider territory. That memo is itself a piece of evidence. Auditors and regulators respect a documented, defensible judgment far more than a confident assertion with nothing behind it.

Which articles become code

This table is the spine of the whole course. Read it as a translation exercise: legal language on the left, an engineering artifact on the right, and the course module where you build it.

ARTICLE(S)	WHAT THE ACT SAYS (PARAPHRASED)	WHAT YOU ACTUALLY BUILD	MODULE
Art 9	Establish and maintain a risk management system across the AI system's lifecycle	A living risk register, not a one-time document; revisited on every material change	M02 / M09
Art 10	Data governance: quality, relevance, representativeness of training/validation/testing data; examine for possible biases	Data quality checks and a real bias evaluation, not a paragraph asserting the data is fine	M03
Art 11 + Annex IV	Draw up technical documentation before the system is placed on the market, keep it current	A documentation pack generated from a model card and the evidence produced by other modules, mapped section by section to Annex IV	M08
Art 12	Enable automatic recording of events ("logs") over the system's lifetime	A logging schema with defined event types, retention, and tamper-evidence (hash chaining)	M06
Art 13	Design for transparency to deployers: instructions for use, known limitations, human oversight measures	Deployer-facing documentation and an API contract that actually surfaces confidence and limitations, not just a PDF	M08
Art 14	Ensure the system can be effectively overseen by natural persons	A human-in-the-loop or human-on-the-loop architecture with measured effectiveness: do overseers actually catch failures, not just a stop button nobody tests	M07
Art 15	Achieve an appropriate level of accuracy, robustness, and cybersecurity	An eval pipeline (accuracy and robustness: out-of-distribution tests, drift monitoring) and a	M04, M05

ARTICLE(S)	WHAT THE ACT SAYS (PARAPHRASED)	WHAT YOU ACTUALLY BUILD	MODULE
		security test suite (adversarial inputs, prompt injection, guardrail-bypass attempts)	
Art 17	Maintain a quality management system, documented as written policies	The process wrapper around all of the above, versioned, so the evidence is reproducible on the next release	M09
Art 26	Deployer obligations: use per instructions, assign competent oversight, keep received logs, monitor for anomalies	The deployer side of the logging and oversight schemas from Articles 12 and 14, plus an intake process for provider instructions	M06, M07
Art 27	Deployers of certain high-risk systems must conduct a Fundamental Rights Impact Assessment before first use	A structured FRIA document, reusable across deployments of the same system with the same use case	M08
Art 72, 73	Post-market monitoring; report serious incidents to market surveillance authorities within fixed deadlines	A CI/CD job that reruns the eval suite and refreshes evidence on every release, plus an incident-reporting runbook wired to the official templates	M09

Each row is, roughly, one module of this course. That is the deal the course makes with you: by the end, every row on the right has been something you personally ran.

The honest timeline

Get this right, because a lot of material circulating in mid-2026 is stale, and stale AI Act timelines cut in both directions: some understate what already binds you, some overstate how urgent the rest is.

DATE	WHAT BINDS	STATUS AS OF MID-2026
2 February 2025	Prohibited practices (Chapter II)	In force
2 August 2025	GPAI model obligations (Articles 51 to 56); the penalties framework (Chapter XII)	In force
2 August 2026	Transparency obligations (Article 50): chatbot disclosure, synthetic-content marking; GPAI-provider fines (Article 101) and the AI Office's full formal enforcement powers	Applies soon, unaffected by the amendment below
2 December 2027	High-risk AI systems under the Annex III, use-case route: the whole of Articles 8 through 15 and the surrounding Chapter III Section 3 obligations	Moved from the original 2 August 2026 date by the 2026 Digital Omnibus amendment
2 August 2028	High-risk AI systems that are safety components of already-regulated products (Annex I route: machinery, medical devices, and similar)	Moved from 2 August 2027

Two precisions matter here, and both are worth saying slowly.

The new dates are fixed calendar dates, not conditional ones. An earlier version of the Commission's simplification proposal would have tied the high-risk dates to a Commission decision confirming that harmonised standards or usable guidance were actually available, with the current dates only as a backstop. The European Parliament and the Council rejected that conditionality during negotiation. What survived is a flat date: 2 December 2027 for Annex III systems, full stop. Do not let anyone tell you the deadline floats with the standards process; it does not, by design.

Only the high-risk dates moved. If someone assumes the whole Act slipped to 2027, they are wrong in a way that can cost real money: prohibitions have applied since

February 2025, GPAI obligations and the general penalties framework since August 2025, and neither was touched by the 2026 amendment. Article 4's AI-literacy duty, softened in wording but not removed, is also unaffected. The amendment is narrow and its name, the "Digital Omnibus," describes its scope accurately: a package of technical simplifications to the AI Act and adjacent digital legislation, not a renegotiation of the Act's substance.

> AS OF MID-2026
>
> One more honesty point, specific to when this chapter was written: the Digital Omnibus reached full political agreement in 2026 (trilogue agreement in May, European Parliament's formal endorsement on 16 June, the Council's final green light on 29 June), but as of early July 2026 it had not yet been published in the Official Journal. That publication is close to a formality at this stage of the process, and the dates above reflect the agreed text, but a course or book that claims a Regulation number and an exact in-force date before the Official Journal actually carries them is guessing. Check the OJ before you cite this to someone who needs precision.

Given all that, the obvious question is why build any of this now, if high-risk obligations do not bind until December 2027. The answer is architectural, not legal. Logging, evaluation pipelines, and structured documentation are cheap to design in on day one and expensive to retrofit into a system that already has users, a data pipeline, and a release cadence. December 2027 is runway. Runway is for building, not for waiting.

Evidence beats certificates, for now

> WATCH OUT
>
> Somewhere in your organization, someone believes that buying an ISO/IEC 42001 certificate makes the company AI Act compliant. This is one of the most expensive misconceptions currently circulating in AI governance circles, and it is worth being precise about why it is wrong.

Article 40 of the Act gives a presumption of conformity to systems built according to harmonised European standards, but only standards that have actually been cited in the Official Journal of the European Union under Regulation (EU) No 1025/2012. As of mid-2026, that citation list for AI Act standards is empty. CEN-CENELEC's Joint Technical Committee 21 is drafting the relevant standards (covering, among other things, quality management, risk management, data governance, bias management, logging and oversight, and accuracy and robustness) under a standardisation request the Commission amended in June 2025 after the original 2025 deadline was missed. The current informal target for the first wave is the second half of 2026. Targets in this space have already slipped once.

ISO/IEC 42001, the international AI management-system standard, is genuinely useful. It gives you a control structure, an auditor-recognized vocabulary, and a certification that signals seriousness to a customer or a board. What it does not give you is the Article 40 presumption of conformity. No ISO standard is a harmonised European standard cited in the Official Journal under the AI Act, and none is scheduled to become one.

So what do you do while the harmonised standards catch up? You do what this course is built around: evidence-first engineering. Every obligation in the table above produces an artifact as a by-product of building the system properly: a classification memo, a bias report, evaluation results as structured data, a security test log, a logging schema, a measured oversight report, a documentation pack. When the harmonised standards eventually land, they will very likely ask for something close to the same evidence, expressed in their own vocabulary. Teams that already have the evidence spend their energy on the mapping exercise. Teams that only have a certificate start from close to zero.

The toolchain

Everything used in this course is open, and none of it is proprietary to the course; you keep all of it.

- **testing-tutorial** carries the largest share of the load: notebooks that walk from manual factual checks through automated evaluation, LLM-as-judge pipelines, and full eval suites, then into security testing, prompt injection, and guardrail-bypass exercises. This is the Article 15 engine for both accuracy and cybersecurity.
- **geobias** runs a real bias study: a panel of evaluators from different backgrounds scoring an open-weight model on the same prompts. It becomes the Article 10 bias-detection exercise, and its report format becomes a template for your own bias evidence.
- **vigil** red-teams a system and then measures, empirically, whether human overseers actually catch the failures it introduces. That measurement is exactly the question Article 14 eventually asks: not "does a human sit in the loop" but "does the human in the loop actually work."
- **warden** tests an LLM-as-judge setup against real, public jailbreak attempts and reports the results honestly, including what got through. It feeds the security module alongside testing-tutorial's own security notebooks.
- **Official artifacts**: the actual templates the Commission has published, including the GPAI Model Documentation Form, the Article 53 training-data-summary template, the Article 73 and Article 55 incident-report templates, and the draft high-risk classification guidelines, which run to roughly one hundred fifty pages of worked examples for the Annex III route alone.
- **The AI Act Service Desk**: the Commission's own interactive Compliance Checker and Act Explorer, which is where the next chapter, on classification, begins.

Worksheet: map your own systems to roles

Before the next chapter walks three fictional systems through the same exercise in full detail, do a first pass on your own work. For three systems you build, maintain, or integrate (they can be shipped products, internal tools, or anything in between), answer:

1. **What does the system do, in one sentence, and who touches its output?** Not what it is built from; what it decides or produces.

2. **Who places it on the market or puts it into service, and under whose name?** This is your first signal for provider versus deployer.
3. **Is any part of it a general-purpose AI model that you trained or substantially modified yourself**, as opposed to one you call through an API or fine-tune in the ordinary sense? If you fine-tuned it, estimate roughly how much compute that took relative to a public estimate of the base model's training run, and write one sentence on why you do or do not think it crosses into "substantial modification" territory.
4. **Has this system, or a close relative of it, ever changed hands**: built for internal use and later sold, licensed, or handed to a different legal entity? If so, note the date that happened, because that is very likely the date your role changed.
5. **For each system, write one line: "I am the __ of this system, because _____."** Leave it blank if you are genuinely unsure. Unsure is a valid, common answer at this stage, and it is exactly what the next chapter's classification tools are for.

Keep what you write. It becomes the first page of your evidence pack.

Appendix: worksheet answer guidance

There is no universal answer key here, because the exercise is about your own systems, not the fictional ones. What follows is guidance on how to check your own reasoning, not a set of correct answers.

- If you answered question 2 with a company name and question 5 with "provider," double-check that the company's name or trademark is actually what appears to the end user or the deploying organization. Providership attaches to whoever's name is on the system, which is not always the team that wrote the code.
- If question 3 produced a fine-tuning job and you are unsure whether it crosses the compute threshold, the honest failure mode is skipping the estimate entirely. Even a rough order-of-magnitude comparison (your fine-tuning run's GPU-hours against a public estimate of the base model's training compute) is worth writing down. The absence of an estimate is worse than an estimate you later revise.

- If question 4 turned up a system that moved from internal use to external distribution, treat the date of that move as the date a new classification exercise should have started, not the date the code changed. The next chapter's classification walkthrough assumes this kind of transition as one of its worked examples.
- If every answer to question 5 was "unsure," that is not a failure of this chapter. It means Chapter 2's classification tools, including the Commission's own Compliance Checker, are about to do real work for you. Bring these same three systems into that chapter.

CHAPTER 2

Is My System in Scope? Classification in Practice

Before this chapter, engineering guidance, not legal advice: this is careful research turned into practice, and for decisions with real legal exposure you bring counsel.

Every obligation in this course is conditional on one answer. Ask “is my system in scope” and you get one of three results, and the three results have wildly different price tags.

Answer one: your system lands in the limited-transparency lane. The workload is small and dated early. Article 50 disclosure and marking duties apply from 2 August 2026: tell people they are talking to a machine, mark AI-generated content where relevant. Days of engineering, not months.

Answer two: your system is outside Annex III and not a safety component under Annex I. The workload is one honest document: a written assessment of why, filed before the system reaches the market, revisited when the system changes.

Answer three: your system is high-risk under Article 6. The workload is the rest of this course: a risk management system, data governance, technical documentation, logging, human oversight, accuracy and robustness testing, conformity assessment, registration, the full Chapter III Section 2 program (Articles 8 to 15). Those duties bind from 2 December 2027. That date is fixed; it is not tied to harmonised standards becoming available, and no later proposal has changed it.

The deliverable of this module is the artifact that answers the question: a classification memo. It is evidence artifact number one, and everything else in this course assumes it already exists.

What Counts as an AI System

Classification starts one level below Article 6, at the definition in Article 3(1):

> THE ACT SAYS · ARTICLE 3(1)
>
> ‘AI system’ means a machine-based system that is designed to operate with varying levels of autonomy and that may exhibit adaptiveness after deployment, and that, for explicit or implicit objectives, infers, from the input it receives, how to generate outputs such as predictions, content, recommendations, or decisions that can influence physical or virtual environments.

Read it slowly, because two words carry the whole test. “May” exhibit adaptiveness: adaptiveness is a possibility, not a requirement. A system that never learns after deployment, that runs the same frozen model forever, is still an AI system if it infers. “Infers”: this is the heart of it. A rules engine that maps inputs to outputs through code you wrote and can fully trace executes a mapping; it does not infer one. A model that was trained or configured to derive its own mapping from data, and that produces an output the developer did not hand-code, is inferring.

The Commission’s definition guidelines, published to accompany Article 3(1), draw the boundary from the other side: what is not covered. Traditional software following fixed, deterministic rules written directly by a human, with no learned or inferred component, sits outside the definition entirely. A spreadsheet formula, a hand-coded decision tree, a keyword-matching filter: none of these are AI systems no matter how consequential their output.

This matters because the edge cases are common in real products. A “smart” form that routes a support ticket by matching against a fixed list of keywords is not an AI system. The same form, once it swaps the keyword list for an embedding-similarity classifier trained on historical tickets, becomes one, even if nobody user-facing notices the difference. Classification runs on what the system does technically, not on the marketing copy around it. Check this first, because everything downstream (Annex I, Annex III, the derogation, the memo) only applies once you have cleared this gate.

Route One: Annex I and Regulated Products

Article 6(1) sets the first of two routes to high-risk, and it runs through product-safety law, not through the AI Act's own use-case list:

> **THE ACT SAYS · ARTICLE 6(1)**
>
> Irrespective of whether an AI system is placed on the market or put into service independently of the products referred to in points (a) and (b), that AI system shall be considered to be high-risk where both of the following conditions are fulfilled: (a) the AI system is intended to be used as a safety component of a product, or the AI system is itself a product, covered by the Union harmonisation legislation listed in Annex I; (b) the product whose safety component pursuant to point (a) is the AI system, or the AI system itself as a product, is required to undergo a third-party conformity assessment, with a view to the placing on the market or the putting into service of that product pursuant to the Union harmonisation legislation listed in Annex I.

Annex I lists the Union harmonisation legislation this route hooks into: machinery, medical devices, toys, lifts, radio equipment, and similar regulated product categories. If your model steers a robot arm, calibrates a medical infusion pump, or sits inside a machine that already needs a notified body's sign-off, this route can catch it regardless of whether the use-case appears anywhere in Annex III.

The 2026 Omnibus amendment narrows this route, and the narrowing matters for anyone building monitoring, optimisation, or advisory software around physical equipment. Two changes, both pending Official Journal publication:

- The definition of "safety component" is narrowed. Systems that assist a user, optimise a process, improve efficiency, or support quality control are excluded from the safety-component category, unless their failure would endanger health or safety directly. A dashboard that recommends maintenance windows is not, by itself, a safety component; a system that actuates a shutdown or overrides a safety interlock plausibly is.
- Machinery moves from Annex I Section A to Section B. The practical effect is a lighter compliance path for machinery-embedded AI that still needs to satisfy the

machinery regulation, without automatically inheriting the heaviest Annex I obligations.

The caption worth remembering: Annex I and Annex III are an OR, not an AND. A system can fail the Annex I test entirely and still be high-risk because it is listed in Annex III, and vice versa. Check both routes every time; do not stop at the first "no."

Route Two: Annex III, Area by Area

Article 6(2) is the second route, and it is a closed list: "In addition to the high-risk AI systems referred to in paragraph 1, AI systems referred to in Annex III shall be considered to be high-risk." No product-safety hook required. If your use case sits in one of the eight areas, you are in scope irrespective of how the system is built.

The eight areas, one line each:

1. **Biometrics** (in so far as use is permitted under Union or national law): remote biometric identification, biometric categorisation by sensitive attributes, and emotion recognition.
2. **Critical infrastructure**: safety components managing critical digital infrastructure, road traffic, or water, gas, heating, or electricity supply.
3. **Education and vocational training**: admission decisions, evaluating learning outcomes, assessing the level of education a person can access, and monitoring or detecting prohibited behaviour of students during tests.
4. **Employment, workers' management, and access to self-employment**: recruitment and candidate evaluation, and decisions on promotion, termination, task allocation, or performance monitoring.
5. **Access to essential private and public services**: eligibility for public assistance benefits, creditworthiness and credit scoring, life and health insurance risk assessment and pricing, and emergency-call classification and dispatch, including emergency healthcare patient triage systems.
6. **Law enforcement** (in so far as permitted): victimisation risk assessment, polygraph-style tools, evidence reliability evaluation, reoffending risk, and profiling

in criminal investigations.

7. **Migration, asylum, and border control** (in so far as permitted): polygraph-style tools, risk assessment of entrants, asylum and visa application support, and detection or identification of natural persons in migration contexts.
8. **Administration of justice and democratic processes**: assisting judicial research and application of law, and systems influencing election outcomes or voting behaviour.

Employment deserves the closest look, because it is the area this course's capstone system lives in. Annex III, point 4(a), reads in full:

> THE ACT SAYS · ANNEX III, POINT 4(A)
>
> AI systems intended to be used for the recruitment or selection of natural persons, in particular to place targeted job advertisements, to analyse and filter job applications, and to evaluate candidates.

That sentence is not close to any line. It is the textbook case, and it names, almost word for word, what an automated CV screener does.

The Article 6(3) Derogation Filter

Landing inside an Annex III area is not automatically the end of the analysis. Article 6(3) carves out a derogation for systems that, despite matching the area, do not pose a significant risk:

> THE ACT SAYS · ARTICLE 6(3)
>
> By derogation from paragraph 2, an AI system referred to in Annex III shall not be considered to be high-risk where it does not pose a significant risk of harm to the health, safety or fundamental rights of natural persons, including by not materially influencing the outcome of decision making.

The derogation is not automatic; it applies only where one of four conditions is met:

1. **Narrow procedural task.** The system performs a bounded, mechanical step, not a substantive judgment. Example: reformatting a document, extracting fields from a form, routing a ticket to the right queue by topic.
2. **Improves the result of a previously completed human activity.** The human already reached a conclusion; the system refines or polishes that output without changing its substance. Example: cleaning up grammar in a human-drafted report.
3. **Detects decision-making patterns or deviations from prior decision-making patterns, without replacing or influencing the previously completed human assessment, without proper human review.** Example: a dashboard that flags that this quarter's approval rate diverged from the historical baseline, purely for a human to investigate later.
4. **Preparatory task to an assessment relevant to an Annex III use case.** Example: pulling together the documents a caseworker will read before that caseworker makes the actual eligibility decision.

Then comes the line that overrides all four: "Notwithstanding the first subparagraph, an AI system referred to in Annex III shall always be considered to be high-risk where the AI system performs profiling of natural persons." Profiling, in the sense the Act imports from data protection law, means automated processing of personal data to evaluate aspects of a person: their performance, behaviour, reliability, preferences, or similar personal characteristics. If your system does that, none of the four conditions rescue it. Profiling kills the derogation outright.

The gotcha worth writing on a sticky note above the monitor: a system that ranks, scores, or filters people based on inferred personal characteristics is very likely profiling, and profiling is an always-high-risk trigger, independent of how narrow or preparatory the task otherwise looks.

Documenting the Decision: Article 6(4) and Registration

Deciding your Annex III system is not high-risk is not the end of the work; it creates a duty of its own. Article 6(4):

> THE ACT SAYS · ARTICLE 6(4)
>
> A provider who considers that an AI system referred to in Annex III is not high-risk shall document its assessment before that system is placed on the market or put into service. Such provider shall be subject to the registration obligation set out in Article 49(2). Upon request of national competent authorities, the provider shall provide the documentation of the assessment.

Three things follow from that one paragraph. First, the documentation has to exist before the system reaches the market, not after a regulator asks for it. Second, even an opt-out from high-risk status triggers a registration duty under Article 49(2), separate from the full registration that applies to confirmed high-risk systems. The 2026 Omnibus amendment lightens this registration burden for self-assessed non-high-risk Annex III systems (two of the information points in Annex VIII Section B are deleted), but it does not remove the obligation, and the amendment is still pending Official Journal publication. Third, the documentation has to be producible on request, which means it has to be findable, not just written once and forgotten in someone's inbox.

This is the operational core of the classification memo this module builds: a written, dated record of the reasoning, kept where a national competent authority could ask for it and get an answer the same day.

The Draft Classification Guidelines

> AS OF MID-2026
>
> Article 6(5) obliged the Commission to publish practical guidelines and a comprehensive list of worked examples, no later than 2 February 2026. The Commission missed that internal date and published a draft on 19 May 2026: three parts, with part three running to 148 pages of worked yes-or-no examples across the Annex III areas. Public consultation runs until 23 July 2026, and the final version is expected by the end of 2026.

Treat the draft as strong indicative guidance, not settled law. It reflects the Commission's current thinking and will very likely track closely onto the final text, but it is not itself binding, and specific examples could still shift before the Commission adopts the final version. The practical habit this module wants you to build: when your case is ambiguous, do not reason from first principles alone. Search the draft's worked examples for the nearest fact pattern, and cite it in your memo. Guideline citations turn a memo from "my opinion" into "here is the closest official example, here is how our system differs, here is why the conclusion follows." That is a materially stronger document to hand a regulator, or to a client's legal team, than reasoning alone.

Three Systems, Three Verdicts

The rest of this chapter classifies three fictional systems in full. They recur through the whole course, so the reasoning here is the reasoning every later module inherits.

BriskDesk: The Chatbot That Never Touches Annex III

Solvana s.r.o., a Prague SaaS shop, builds BriskDesk: a customer-support chatbot embedded as a widget on European e-shops. It answers product questions, looks up order status, and starts returns, using retrieval over each client's catalog and help-center content, plus tool calls into the client's order API. The underlying model is reached through a hosted API; Solvana does not train or fine-tune it.

Walk the tree. Is it an AI system? Yes: it infers a response from free-text input using a model whose mapping was learned, not hand-coded. Is Solvana a GPAI model provider,

with model-level obligations like training-data summaries? No: Solvana calls someone else's model through an API. That makes Solvana the provider of an AI system, not the provider of the underlying general-purpose model. This is the same fine-tuning-trap logic from Module 1: calling, prompting, and retrieval-augmenting a frontier model does not make you its provider.

Does it fall in Annex III? Walk the eight areas: no biometrics, no critical infrastructure, no education, no employment decisions about a specific worker, no essential-service eligibility or credit or insurance, no law enforcement, no migration, no justice or democratic process. Customer support for an e-shop is not listed anywhere in Annex III. Does the Annex I product-safety route apply? No; BriskDesk is not a safety component of any regulated product.

Result: limited-risk lane. BriskDesk carries Article 50(1) disclosure (the person chatting has to know they are talking to a machine) and, where it generates content, Article 50(2) marking. Both duties apply from 2 August 2026, which is earlier than the high-risk timeline. The lesson this system carries through the course: the light lane is dated sooner than the heavy lane, and "light" does not mean "no engineering work." Solvana is the provider; each e-shop that embeds the widget is a deployer.

VibraSense: Not High-Risk, Until It Is

Norrfelt Industrial AB, a Swedish pump manufacturer, builds VibraSense: an in-house model that consumes vibration and temperature sensor time series from the pumps it services and outputs a failure-risk score. That score schedules maintenance visits and alerts a human service-planning team. It does not control or shut down any machine.

Annex III? No area covers predictive maintenance scheduling; it is not biometrics, not critical infrastructure in the sense the Annex uses (VibraSense is advisory software watching sensors, not a safety component managing the infrastructure itself), not any of the other six areas. Annex I? This is where the 2026 narrowing bites directly. VibraSense is not a safety component of the pump: it optimises maintenance scheduling and alerts humans, and its failure does not directly endanger health or safety, because it never actuates anything. The Omnibus amendment's narrower safety-component

definition draws exactly this line: user-assistance and optimisation software is excluded from the safety-component category unless its failure would endanger health or safety directly.

Result: not high-risk. Norrfelt documents the assessment under Article 6(4) and keeps the memo. The honest nuance, and the reason this system exists in this course: the answer flips the moment a future version is wired to auto-derate or stop the pump based on its own output, because that removes the "never actuates" fact the whole conclusion rests on. Classification runs on intended purpose, and intended purpose is not fixed once and forgotten; it has to be revisited whenever the system's actual function changes.

HireSift: The Textbook Case

Arbeta GmbH, an HR-tech vendor in Vienna, builds HireSift: a module inside applicant-tracking systems used by mid-size employers. It parses CVs, extracts features (skills, experience, education), scores and ranks candidates against a job profile using a model trained on ten years of historical hiring outcomes, filters out the bottom of the pool, and writes short shortlist summaries with an LLM.

Annex III? Direct hit, point 4(a): "recruitment or selection of natural persons… to analyse and filter job applications, and to evaluate candidates." HireSift does exactly that, in almost the same words as the Annex.

Try the derogation. Narrow procedural task? No: ranking and filtering candidates is a substantive judgment about people's employment prospects, not a mechanical step. Improving a previously completed human activity? No: nothing has been completed yet when HireSift filters the applicant pool; it is making the primary judgment, not polishing one. Detecting deviations from prior decision patterns without influencing them? No: HireSift's output directly determines who advances, which is the opposite of "without influencing." Preparatory task to an assessment? Arguably closer, but the facts do not support it either: HireSift filters out candidates before any human sees them, which is deciding, not preparing. And even setting the four conditions aside, HireSift scores and ranks individuals based on inferred personal characteristics drawn from

their history, which is profiling of natural persons. Profiling forecloses the derogation regardless of which of the four conditions might otherwise apply.

Two independent locks, either one sufficient on its own: it materially influences the hiring outcome, and it profiles natural persons. Result: high-risk. The full Chapter III Section 2 program, Articles 8 through 15, binds Arbeta from 2 December 2027. Arbeta is the provider; the employers who deploy HireSift inside their applicant-tracking system are deployers. Every later module in this course builds a piece of HireSift's evidence pack: bias examination over the historical training data, declared accuracy and robustness metrics, a logging schema, measured human oversight, and a technical documentation skeleton under Annex IV.

The Memo as Living Evidence

The output of this module is a file, not an opinion: `01_classification_memo.md`, built from a fixed skeleton so every system in your portfolio produces a comparable document.

```
01_classification_memo.md
├── System description and intended purpose
├── AI-system test (Art 3(1)): infers vs executes, edge-case notes
├── Route analysis
│   ├── Annex I: safety-component check, conformity-assessment check
│   └── Annex III: which area, if any, and why
├── Derogation analysis (Art 6(3), all four conditions)
├── Profiling check (overrides the derogation if positive)
├── Conclusion: high-risk / limited-risk lane / out of scope
├── Role: provider or deployer, and for whom
├── Guideline examples cited (draft Art 6(5) worked examples, with page refs)
└── Review triggers
```

The last section is the one people skip and the one that matters most. A classification memo is not a certificate issued once; it is a snapshot of a conclusion that depends on facts, and facts change. Three triggers should force a re-read of the memo, every time:

- **Purpose change.** The system starts being used for something its original intended purpose did not cover. A scheduling tool that starts influencing who gets seen first is not the same system it was on day one.
- **Feature change.** A capability is added that changes which facts the conclusion rested on. VibraSense stays "not high-risk" exactly as long as it never actuates anything; the day it does, the memo is wrong until someone rewrites it.
- **New deployment context.** The same system, deployed by a different kind of customer or into a different regulatory environment, can land in a different Annex III area, or a different role split between provider and deployer.

Write the memo before someone asks for it. Revisit it when the purpose moves.

Worksheet: Classify Two Ambiguous Systems

Using the memo skeleton above, classify the following two systems. Both are deliberately closer to the line than BriskDesk, VibraSense, or HireSift. Write out the AI-system test, the route analysis, the derogation analysis, the profiling check, and your conclusion, before reading the appendix.

System A: the clinic triage-support chatbot. A small private outpatient clinic deploys a web chatbot. Patients describe symptoms in free text. The chatbot, built on an LLM with a symptom-classification prompt, asks follow-up questions and produces one of three outcomes: it books a same-day appointment slot, it queues a short triage note for a nurse to review within the hour, or, when it detects a defined set of red-flag symptoms, it immediately displays a message telling the patient to call emergency services, without any human reviewing that message first.

System B: the exam-proctoring add-on. An ed-tech vendor sells a browser add-on that online course platforms embed during timed exams. Using the webcam and microphone feed plus keystroke and mouse telemetry, it scores the likelihood of prohibited behaviour (gaze aversion, a second voice in the room, tab-switching, atypical typing cadence) for each student in real time. Flagged sessions surface to a human proctor watching a dashboard, who decides whether to intervene during the exam;

every flagged session is also stored for the institution's academic-integrity committee to review afterward.

Appendix: Reasoned Answers

System A: the clinic triage-support chatbot.

AI-system test: yes. The urgency label is inferred by an LLM from unstructured symptom text, not produced by a fixed rule table.

Route analysis: no Annex I hook; a clinic chatbot is not a safety component of a regulated product. Annex III: the closest area is point 5(d), which lists systems intended to evaluate and classify emergency calls, to dispatch or establish priority in dispatching emergency first response services, "as well as of emergency healthcare patient triage systems." The chatbot does not literally dispatch an ambulance; it evaluates urgency and, in one of its three outcomes, directly instructs a patient toward emergency care. Annex III areas are defined by use case, not by deployment setting, so a private clinic's front door is not automatically excluded just because it is not a hospital emergency department.

The system's three outcomes need separating, because they carry different derogation prospects. The same-day-appointment and nurse-queue outcomes route the patient toward a human decision before anything consequential happens to them: a nurse still reviews the triage note within the hour. That looks like a genuine candidate for the Article 6(3)(d) derogation, "a preparatory task to an assessment relevant for the purposes of the use cases listed in Annex III", provided the nurse review is real and happens before, not after, any action is taken on the patient. The immediate red-flag outcome does not fit any of the four conditions: it is not narrow-procedural (classifying a medical emergency is a substantive judgment), it does not improve a completed human activity (no human has acted yet), it is not detecting deviations from a prior decision pattern, and it is not preparatory, because the patient receives the instruction directly, with no human reviewing it first.

Profiling check: the system evaluates a specific patient's health-related behaviour and symptoms to produce an individualised urgency judgment. That fits the profiling

definition regardless of which outcome path fires.

Conclusion: split the system, or classify conservatively. If the same chatbot both queues low-urgency cases for nurse review and autonomously instructs high-urgency cases without review, the safer, honest reading treats the whole system as high-risk under Annex III 5(d): the profiling override applies across the system, and the one outcome path without human review cannot be rescued by the derogation. The clinic (or the vendor, depending on who places it on the market) should either add a mandatory human check before the emergency-instruction path fires, which strengthens the case for treating that path as a preparatory task, or accept the high-risk classification and build the Chapter III Section 2 program around it. This is exactly the kind of case where citing the nearest worked example from the draft Article 6(5) guidelines, once the final version is out, turns a judgment call into a documented, defensible position.

System B: the exam-proctoring add-on.

AI-system test: yes. Behavioural scores are inferred from video, audio, and telemetry, not matched against a fixed rule.

Route analysis: no Annex I hook. Annex III, point 3(d), is a direct textual match: “AI systems intended to be used for monitoring and detecting prohibited behaviour of students during tests in the context of or within educational and vocational training institutions at all levels.” Real-time exam-integrity monitoring is precisely what this clause names.

Derogation: none of the four conditions fit cleanly. It is not a narrow procedural task, because flagging likely cheating is a substantive judgment about a student, not a mechanical step. It cannot improve a previously completed human activity, because there is no completed human judgment yet when the flag is raised mid-exam. Condition (c), detecting deviations from prior decision-making patterns without influencing a previously completed assessment, does not fit either: there is no “previously completed” human assessment during a live exam session for the system to leave undisturbed; it is generating the primary signal a proctor acts on in real time. It is not a preparatory task to an assessment in the Article 6(3)(d) sense, because the flag is the assessment output itself, not input gathered ahead of one.

Profiling check: the system evaluates a named student's behaviour, in real time, to predict the likelihood of a specific personal characteristic (dishonesty). That is profiling of natural persons in the sense the Act uses the term, and it locks in the always-high-risk override independently of the derogation analysis above.

Conclusion: high-risk, on two independent grounds, the same double-lock pattern as HireSift: a direct Annex III hit with no available derogation, plus profiling that would override the derogation even if one had otherwise applied. The vendor is the provider; each institution that embeds the add-on during its exams is a deployer, with duties including informing students, under Article 26(11), that they are subject to the system, and assigning human oversight to proctors with the competence and authority to intervene.

CHAPTER 3

Data Governance and Bias: Engineering Article 10

Module 2 told you whether your system is in scope. If it is high-risk and it learns from data, Article 10 is the next thing an auditor opens. It binds from 2 December 2027, alongside the rest of Chapter III Section 2. This module treats it as what it actually is: a specification for a datasheet, a bias-examination workflow, and an evidence folder, not a vague call for "good data."

The anchor sentence for the whole module: if you train an AI model with data, your training, validation, and testing sets carry quality duties, and you must examine and mitigate bias in them. If you train nothing, the duty narrows but does not disappear. BriskDesk, Solvana's customer-support chatbot, calls a hosted frontier model and trains nothing itself, so most of Article 10 does not bind it directly. But BriskDesk still has a de facto data governance problem: its RAG corpus of product catalog entries and help-center articles is the data the system actually reasons over, and a stale or unrepresentative corpus produces the same kind of harm a bad training set would. HireSift, Arbeta's CV screener, is squarely inside Article 10: it trains a ranking model on ten years of historical hiring outcomes, exactly the scenario the article was written for. This chapter uses HireSift as the running example and returns to BriskDesk where Article 10(6) narrows the scope for non-trained systems.

Article 10, clause by clause, with the artifact map

Article 10 is titled "Data and data governance." Read it once as a legal text and it sounds abstract. Read it as a checklist for a document you already know how to write, a dataset datasheet, and it becomes concrete.

10(1): when the article applies

The article opens by scoping itself to systems that learn from data:

> THE ACT SAYS · ARTICLE 10(1)
>
> High-risk AI systems which make use of techniques involving the training of AI models with data shall be developed on the basis of training, validation and testing data sets that meet the quality criteria referred to in paragraphs 2 to 5 whenever such data sets are used.

The scoping matters. Article 10 is a duty about training, validation, and testing data sets. A system that does not train a model, such as one built entirely on a hosted LLM with no fine-tuning, sits outside most of this article, which is exactly where 10(6) picks the thread back up later.

10(2): eight practices, mapped to artifacts

Paragraph 2 lists the "data governance and management practices" the training, validation, and testing sets must be subject to. Read straight through, each letter maps cleanly onto something a data or ML team already produces, or should:

ART 10(2) PRACTICE	TEXT (PARAPHRASED)	ARTIFACT
(a)	relevant design choices	design record / ADR
(b)	collection process and origin of data (and, for personal data, the original purpose of collection)	datasheet section
(c)	data-preparation operations: annotation, labelling, cleaning, updating, enrichment, aggregation	pipeline code and logs
(d)	the assumptions made about what the data is supposed to measure and represent	datasheet
(e)	assessment of availability, quantity, and suitability of the needed data	data audit
(f)	examination for biases likely to affect health, safety, fundamental rights, or lead to discrimination, especially	bias report

ART 10(2) PRACTICE	TEXT (PARAPHRASED)	ARTIFACT
	where data outputs feed future operations	
(g)	measures to detect, prevent, and mitigate the biases found under (f)	mitigation log
(h)	identification of data gaps or shortcomings and how to address them	known-limitations section

That table is the spine of this module. Everything from here on is about filling in those eight cells honestly, with evidence, for a real system.

10(3): representative, best effort, not perfect

This is the clause developers most often misread, usually by assuming it demands perfect data. It does not. The text:

> THE ACT SAYS · ARTICLE 10(3)
>
> Training, validation and testing data sets shall be relevant, sufficiently representative, and to the best extent possible, free of errors and complete in view of the intended purpose. They shall have the appropriate statistical properties, including, where applicable, as regards the persons or groups of persons in relation to whom the high-risk AI system is intended to be used. Those characteristics of the data sets may be met at the level of individual data sets or at the level of a combination thereof.

Three honest readings worth pulling apart:

- **“Sufficiently representative”** is a standard, not a percentage. It asks whether the data resembles the population the system will actually act on, not the population the data happened to be collected from.

- **“To the best extent possible, free of errors and complete”** is a best-effort clause with the qualifier built into the sentence. It does not require error-free or complete data; it requires documented, good-faith effort toward that goal. A dataset with known, documented gaps is not automatically non-compliant. A dataset with undocumented, unexamined gaps is.
- **“Appropriate statistical properties… as regards the persons or groups of persons”** is where fairness enters the data-quality clause directly, not just through the bias-examination duty in 10(2)(f). A dataset can be statistically well-behaved in aggregate and still fail this test for a specific subgroup the system is intended to serve. And the closing sentence, that these characteristics may be met “at the level of individual data sets or… a combination thereof,” gives an engineering escape hatch: no single source dataset needs to be independently representative, as long as the combined corpus is.

Representative of what, exactly? Of the intended-purpose population. For HireSift, that is the applicant pool the system will actually screen once deployed: the mix of candidates, roles, and countries its employer clients draw from, not the ten years of historical hiring data that happened to be sitting in Arbeta’s client databases. Those two populations can diverge badly, and the gap between them is precisely what 10(3) and 10(4) ask you to examine.

10(4): setting matters

> THE ACT SAYS · ARTICLE 10(4)
>
> Data sets shall take into account, to the extent required by the intended purpose, the characteristics or elements that are particular to the specific geographical, contextual, behavioural or functional setting within which the high-risk AI system is intended to be used.

HireSift trained its ranking model on ten years of one country’s hiring data. If Arbeta then sells HireSift into a different national market, with a different labor law regime, different CV conventions, and different signals of qualification, the geographic and

behavioural setting has shifted under the model. That is a legal issue under Article 10(4), not only a modeling accuracy issue for Module 4. VibraSense gives the inverse illustration outside the high-risk lane: a predictive-maintenance model trained on Norrfelt's fleet of pumps in temperate climates faces a different functional setting when deployed on pumps at an Arctic site, where vibration and temperature baselines differ. The lesson generalizes: "representative" and "appropriate statistical properties" are always relative to a deployment setting, and that setting can change after the model ships.

10(5): the special-category paradox and the 2026 amendment

Paragraph 2(f) requires bias examination for effects on fundamental rights and discrimination. But examining whether a model discriminates on, say, racial or ethnic origin, or disability, or sexual orientation, often requires knowing those attributes for at least a test population, and those are special categories of personal data under the GDPR, ordinarily processed only under narrow conditions. Article 10(5) resolves the paradox with an exceptional, narrow basis:

> THE ACT SAYS · ARTICLE 10(5)
>
> To the extent that it is strictly necessary for the purpose of ensuring bias detection and correction in relation to the high-risk AI systems in accordance with paragraph (2), points (f) and (g) of this Article, the providers of such systems may exceptionally process special categories of personal data, subject to appropriate safeguards for the fundamental rights and freedoms of natural persons.

The word "exceptionally" is doing real work. This is not a general license to collect protected-attribute data; it is a narrow carve-out, and it comes with six cumulative conditions, all of which must hold (paraphrased, with the operative phrase quoted):

1. **Necessity.** The bias work "cannot be effectively fulfilled by processing other data, including synthetic or anonymised data."
2. **Technical limitation and security.** The data is subject to "technical limitations on the re-use," plus "state-of-the-art security and privacy-preserving measures,

including pseudonymisation."

3. **Access control.** Access is secured, safeguarded, and logged, with "only authorised persons" holding "appropriate confidentiality obligations."
4. **No transfer.** The data is "not to be transmitted, transferred or otherwise accessed by other parties."
5. **Deletion.** The data is "deleted once the bias has been corrected or [it] has reached the end of its retention period, whichever comes first."
6. **Documented necessity.** Processing records "include the reasons why the processing… was strictly necessary… and why that objective could not be achieved by processing other data."

HireSift's bias examination may need to know the gender, age band, or inferred ethnic origin associated with historical applicants, to test whether the ranking model treats comparable candidates comparably. Condition 4 is the one developers most often get wrong in practice: this special-category data cannot be handed to the deploying employer, cannot flow into a shared analytics warehouse, and cannot leave the bias-testing process it was collected for. It exists to be examined, not to be operationalized elsewhere.

> AS OF MID-2026
>
> As drafted, Article 10(5) applied only to providers of high-risk AI systems. A 2026 Digital Omnibus amendment, at the time of writing not yet published in the Official Journal, extends this special-category basis to all AI systems and GPAI models, under the same strict-necessity conditions. Read that as a practical win for fairness testing generally: teams building limited-risk systems like BriskDesk gain the same narrow legal basis to test for bias in their outputs, without waiting for a high-risk classification to unlock it. Treat "pending OJ" as exactly that: confirm the citation before relying on it operationally.

10(6): when you train nothing

> THE ACT SAYS · ARTICLE 10(6)
>
> For the development of high-risk AI systems not using techniques involving the training of AI models, paragraphs 2 to 5 apply only to the testing data sets.

This clause narrows, rather than removes, Article 10 for systems built on a pre-trained model the provider does not fine-tune. If HireSift's summarizer used a hosted LLM purely for shortlist notes with no fine-tuning, the practices in 10(2) through 10(5) would still apply, but only to the data used to test that component, since there is no training set. BriskDesk sits in the same place: 10(6) is the clause that actually governs it, and its testing data, the prompts and expected behaviors used to validate the chatbot before and after release, inherits the same quality and bias duties a training set would carry. Its RAG corpus is a related but distinct concern, not a training set in the Article 10 sense, but a governance problem all the same, and one the datasheet discipline below applies to just as usefully.

Datasheets: the practical vehicle

Article 10(2) reads like a specification for a document that already exists in the ML community: the datasheet, proposed by Timnit Gebru and colleagues in "Datasheets for Datasets" (arXiv preprint 2018, published in Communications of the ACM in 2021). The paper's pitch was that every dataset should ship with a structured set of questions and answers covering motivation, composition, collection process, preprocessing, uses, distribution, and maintenance, modeled loosely on the datasheets that accompany electronic components.

The fit to Article 10(2) is close enough that a team can use one document to satisfy both. The datasheet's "collection process" section answers 10(2)(b). Its "composition" and "preprocessing" sections answer 10(2)(c) and (d). Its "uses" section, especially the questions about what the dataset should and should not be used for, is where 10(2)(e) and 10(2)(h) naturally live. What the datasheet format does not cover on its own is the bias examination and mitigation loop of 10(2)(f) and (g); that needs a dedicated bias-report section, which the rest of this chapter builds.

For HireSift, a datasheet answers, for each of the ten years of historical hiring outcomes folded into training: who collected this data and why, what assumptions the "hired" versus "not hired" label encodes, what populations are under-represented, and what the data cannot tell you. That last question, 10(2)(h), is usually the most useful section in the whole document, because it is where a provider admits, on the record, what it does not know about its own training data.

Bias taxonomy for builders

Article 10(2)(f) asks for "examination in view of possible biases." A useful taxonomy, adapted from the fairness-in-ML literature, gives that examination a checklist rather than a vague intention. Six categories, each with a HireSift-shaped example:

- **Historical bias.** The label itself encodes a biased past. HireSift's ten years of "who got hired" data reflects the judgment of human recruiters at client companies, who may themselves have discriminated, consciously or not, against candidates with employment gaps or non-domestic-sounding names. A model trained on those outcomes learns "who historically got hired," not "who is qualified."
- **Sampling and representation bias.** If the ten years of data come overwhelmingly from a handful of client companies in one country and one industry, the training set under-represents the applicant pool HireSift is sold to screen elsewhere: the 10(3) representativeness question, seen through a bias lens.
- **Label bias.** Who defined "success" in the first place. If the training label is a later performance-review score rather than the hiring decision itself, and those reviews carry their own rater bias, the label bias compounds on top of the historical bias in who got hired at all.
- **Measurement and proxy bias.** A model does not need an explicit "ethnicity" field to discriminate. Address or postcode can proxy for socioeconomic status. Employment gaps can proxy for care responsibilities, disability, or gender. A name can proxy for national or ethnic origin. HireSift's feature set, built from parsed CVs, is full of exactly these proxies, and none need to be removed from the CV to remain predictive of a protected attribute.

- **Aggregation bias.** Fitting one global ranking model across heterogeneous subgroups can wash out group-specific signal. What predicts success in an engineering role may differ from what predicts success in a sales role, and pooling both into one model can systematically under-serve whichever subgroup deviates from the dominant pattern in the mix.
- **Feedback loops.** Article 10(2)(f) explicitly flags examination "especially where data outputs influence inputs for future operations." HireSift's shortlisting decisions become next year's "hired" labels at the same client companies, so whatever pattern the model learned this year gets reinforced, not corrected, next year, unless someone actively breaks the loop. This is also where Article 10 meets the accuracy and robustness obligations of Article 15, covered in Module 4: a feedback loop is as much a drift problem as a fairness problem.

Measuring group fairness: a worked example

Bias examination under 10(2)(f) eventually needs numbers, not only categories. The standard toolkit is a handful of group fairness metrics, each answering a different question about a confusion matrix computed separately per group.

METRIC	QUESTION IT ANSWERS	USE WHEN
Selection-rate parity (demographic parity)	Are groups shortlisted at the same rate, regardless of ground truth?	Representation in outcomes is the primary concern, e.g. regulatory or works-council scrutiny of raw shortlist composition
Equal opportunity (TPR parity)	Among genuinely qualified candidates, are groups shortlisted at the same rate?	False negatives, a qualified candidate never seen, are the dominant harm
Equalized odds (TPR and FPR parity)	Do groups have equal true-positive and false-positive rates?	Both false negatives and false positives carry comparable harm
Calibration within groups	Does a given score mean the same thing, in terms of	Downstream humans read the score as a probability and compare it across candidates

METRIC	QUESTION IT ANSWERS	USE WHEN
	actual outcome likelihood, across groups?	

Here is why a single metric will not settle the question, using small, consistent numbers built the way a real audit would compute them. Suppose HireSift's historical test set has 100 candidates in Group A and 100 in Group B, and, importantly, the same underlying qualification rate in both groups: 40 out of 100 are labeled qualified by the historical ground truth.

Group A (100 candidates, 40 actually qualified):

	PREDICTED QUALIFIED	PREDICTED NOT QUALIFIED	TOTAL
Actually qualified	32 (TP)	8 (FN)	40
Actually not qualified	18 (FP)	42 (TN)	60

Group B (100 candidates, 40 actually qualified):

	PREDICTED QUALIFIED	PREDICTED NOT QUALIFIED	TOTAL
Actually qualified	18 (TP)	22 (FN)	40
Actually not qualified	12 (FP)	48 (TN)	60

From these two tables:

- **Selection rate**: Group A shortlists 50 of 100 (0.50); Group B shortlists 30 of 100 (0.30). Fails parity by 20 points.
- **Equal opportunity (TPR)**: Group A's true positive rate is 32/40 = 0.80; Group B's is 18/40 = 0.45. A qualified candidate in Group B is shortlisted at barely half the

rate of an equally qualified candidate in Group A, the largest gap of the four metrics, and the most consequential one for a screening tool.

- **Equalized odds**: the TPR gap above already fails this test; the false positive rate gap adds to it, 18/60 = 0.30 for Group A versus 12/60 = 0.20 for Group B.
- **Calibration (precision, as a proxy for what a score means in practice)**: 32/50 = 0.64 for Group A versus 18/30 = 0.60 for Group B, four points apart, far closer than the other three metrics.

That last line is the point of the exercise. By calibration, the model looks nearly fair: a shortlisted candidate in either group is qualified about six times in ten. By selection rate, equal opportunity, and equalized odds, the same model looks starkly unfair. This is a well-documented mathematical property of these metrics: outside special cases, they cannot all be satisfied simultaneously when base rates or model behavior differ across groups. Article 10 does not ask you to resolve that tension by picking the metric that looks best. It asks you to examine, under 10(2)(f), pick and justify a metric appropriate to the harm model, and document the choice and the mitigation taken under 10(2)(g). That documentation, why equal opportunity over demographic parity, for instance, because false negatives are the harm that matters here, is itself the evidence artifact an auditor reads.

Behavior-level bias testing for LLM components

HireSift's ranking model is trained on tabular data and produces the group fairness picture above. Its summarizer, and BriskDesk's entire chatbot, are built on LLM components the provider did not train. Article 10(2)(f) and (g) still apply to whatever those systems are tested with, per 10(6), and the practical question shifts from "does the training data carry bias" to "does the composed system, prompts, retrieval corpus, tool wiring, behave consistently across comparable inputs."

That is a behavioral question, testable without touching model weights. Hold the task constant and vary only the identity-carrying part of the input: does a support query get the same quality of answer and tone when the customer's name reads as one national origin versus another; does a shortlist summary describe two equivalently qualified

candidates in language that carries a different connotation. The method is a perturbation test: change one variable that should be irrelevant to the task, and measure whether the output changes. This is the trick the geobias study below is built on, applied there to a different comparable-treatment question, and it transfers directly to HireSift's summarizer and BriskDesk's chat responses.

Case study: the geobias evaluator panel

The geobias project runs exactly this kind of behavior-level examination on open-weight LLMs, and its design is worth stealing wholesale for an Article 10(2)(f) evidence pack.

The design: five open-weight models answer prompts across seven test categories built to surface geopolitically loaded framing: narrative framing, refusal asymmetry, entity sentiment, language quality, regulatory values, self-awareness, and historical narrative. The five models themselves span three declared origins: Qwen3 8B and Qwen3.5 9B (China), Llama 3.1 8B and Phi-4 (United States), and Ministral 8B (the European Union). Every response is then scored not by one judge but by a panel of three evaluator models, chosen to span the same three origins: Qwen3 235B (China), Llama 3.3 70B (United States), and Mistral Large 3 (the European Union).

The panel is the trick worth stealing. A single LLM judge inherits a single worldview, and using it to score a bias study risks measuring the judge's bias instead of, or on top of, the bias under test. A panel of evaluators from different origins turns the disagreement between judges into a finding in its own right, not noise to average away. In one published cached run of this design (a 7B-scale sweep, 8 test cases per category, fully reproducible offline from the published `geobias-results` artifacts), the evaluator-disagreement numbers illustrate exactly this. The Mistral Large 3 evaluator (EU origin) scored EU-family models 8.94 on average versus 8.30 for models of other origins, a same-origin leniency gap of about 0.64 points. The Llama 3.3 70B evaluator (US origin) showed a smaller same-origin leniency of about 0.23. The Qwen3 235B evaluator (China origin) showed the opposite pattern, scoring same-origin models 0.28 points lower than other-origin models. (These are the values the module notebook computes from the run's raw evaluation files with the corrected aggregation from the

July 2026 geobias audit; the numbers you reproduce should match it.) Treat that as a small-sample, single-run result worth flagging, not a settled claim about any evaluator's general behavior: 8 cases per category demonstrates the method; it does not license generalizing about population-level tendencies. That caveat is itself part of doing this kind of examination honestly.

The same run's model-side asymmetry numbers show the same shape of finding on the models under test. Qwen3 8B scored its own-region-tagged prompts noticeably harsher than other-region prompts (own-region average 4.61 versus other-region average 7.89, an asymmetry of 3.3 points, on eight own-region and thirteen other-region cases), while Llama 3.1 8B was close to symmetric (asymmetry of minus 0.05). Whatever the underlying cause, the asymmetry itself is measurable, reproducible from the published files, and directly analogous to the "does the system treat comparable inputs comparably" question HireSift and BriskDesk both need answered for their own LLM components.

Every artifact behind these numbers, `config.json`, `test_cases.json`, `metrics.json`, `evaluations.json`, the per-evaluator raw judgment files, and the five model response files, ships in a self-contained cached run, and the full HTML report regenerates from it with no API key and no network call. That offline reproducibility is itself worth noting as a compliance property: an examination an auditor can rerun from the evidence files is a stronger artifact than a one-time report nobody can regenerate.

From bias report to Article 10 evidence

A bias examination is not evidence until it is written down in a form an auditor can read without a walkthrough. The evidence section this module builds toward needs five parts:

1. **Scope and method.** What was examined (which system, which component, which data or behavior), and why this method was chosen over alternatives (for HireSift's ranking model: which groups, which metric, and why; for its summarizer: which perturbation test, over which input variable).

2. **Findings, with numbers.** Not “bias was found” but the actual confusion-matrix numbers, asymmetry scores, or evaluator-disagreement figures, the same shape as the worked example and the geobias numbers above.
3. **Mitigations taken**, satisfying 10(2)(g). What changed as a result: a reweighted training set, a removed proxy feature, a threshold adjusted per group, a prompt rewritten, a guardrail added.
4. **Residual risks and gaps**, satisfying 10(2)(h). What the examination could not rule out, what data was missing, what population was not covered by the test set.
5. **Review cadence.** When this examination gets rerun, and what triggers an earlier rerun (a new client market, a retraining cycle, a detected feedback-loop symptom from 10(2)(f)).

HireSift’s version stitches together three pieces: an outcome test on the historical training data (the confusion-matrix exercise above, run for real on the actual training set), a proxy audit (checking which CV features correlate with protected attributes even without an explicit field for them), and subgroup metrics pulled from the evaluation pipeline Module 4 builds. None of the three alone is the evidence section; written up against the five parts above, together they are.

Worksheet: draft HireSift’s bias-examination plan

Before checking the sample answer in the appendix, draft a bias-examination plan for HireSift’s ranking model, covering four things:

1. **Protected attributes and proxies table.** List the protected attributes plausibly at risk for a hiring tool under EU non-discrimination law, and, for each, the CV or ATS features that could act as a proxy for it.
2. **Metric choice and justification.** Pick a primary group fairness metric for the shortlisting decision and justify the choice against the harm model, using the table from the fairness-metrics section above.
3. **Data needed.** What data the examination requires, including whether any of it counts as special-category personal data under Article 10(5), and for what purpose specifically.

4. **Article 10(5) checklist.** If special-category data is needed, walk through the six cumulative conditions and note, for each, how HireSift's process would satisfy it.

Appendix: sample answer

1. Protected attributes and proxies.

PROTECTED ATTRIBUTE	PLAUSIBLE PROXY IN HIRESIFT'S FEATURE SET
Sex or gender	Given name, career-gap pattern tied to parental leave, gender-coded language in self-written summaries
Age	Graduation year, years of experience, career start date
Race or ethnic origin	Family name, address or postcode, school or university name
Disability	Employment gaps, part-time work history
National origin	Address, language of previous employers, name

2. Metric choice and justification. Equal opportunity (true-positive rate parity) as the primary metric. HireSift's dominant harm is a qualified candidate filtered out before a human ever sees them, which is precisely a false negative in the ranking model's confusion matrix. Selection-rate parity is tracked as a secondary, monitoring-only metric, because employer clients and works councils are likely to ask about raw shortlist composition regardless of the primary metric chosen, and a large unexplained gap there is worth investigating even if equal opportunity holds.

3. Data needed. The historical training set, subgroup-labeled for the protected attributes above wherever inferable; a held-out test set drawn from the intended-purpose population (the applicant pools of the specific employer clients HireSift will actually serve, not only the historical training population); and, for attributes not visible from existing CV fields (inferred ethnic origin or disability status), a purpose-built

annotated sample used only for the bias test, which does qualify as special-category personal data.

4. Article 10(5) checklist for that purpose-built sample.

- Necessity: document why synthetic or anonymised substitutes cannot stand in, specifically that ethnic-origin or disability inference cannot be reliably simulated without real annotated examples.
- Technical limitation and security: pseudonymise the sample; restrict it to a locked-down analysis environment with no export path.
- Access control: name the individuals authorised to access it and log every access.
- No transfer: the sample never reaches the employer-client deployers and never enters Arbeta's general analytics warehouse.
- Deletion: delete the sample once the bias examination is complete, or at the end of its retention period, whichever comes first.
- Documented necessity: record, in the processing-activity records, why this processing was strictly necessary and why no other data would have served the purpose.

That checklist, filled in with real names, dates, and access logs rather than placeholders, is the artifact an auditor will actually ask to see.

CHAPTER 4

Accuracy and Robustness: Article 15 as an Engineering Spec

Module 3 asked whether your training data was governed and your model was examined for bias. This module asks a plainer question: does the system work, and does it keep working. Article 15 of the AI Act answers that question for high-risk systems, and it answers it in a way that should feel familiar to anyone who has shipped a model to production. It does not hand you a pass mark. It asks you to pick your own pass mark, write it down, and keep proving it.

That last part is the one developers underestimate. Article 15(1) closes with a phrase that reframes the whole article: performance must hold "throughout the lifecycle." A single green run in CI, the week before the conformity assessment, is not what this article asks for. It asks for a testing and monitoring habit that survives the system's entire time on the market. This chapter builds that habit in four layers: what the article actually requires, how you choose and declare metrics for a real system, how an evaluation practice matures from a spot check to a pipeline, and how robustness testing catches the failure modes that only show up after release.

Reading Article 15(1) to (4) as a spec

Here is the text, unmodified, from Chapter III Section 2 of the enacted Act:

THE ACT SAYS · ARTICLE 15(1) TO (4)

1. High-risk AI systems shall be designed and developed in such a way that they achieve an appropriate level of accuracy, robustness, and cybersecurity, and that they perform consistently in those respects throughout their lifecycle.
2. To address the technical aspects of how to measure the appropriate levels of accuracy and robustness set out in paragraph 1 and any other relevant performance metrics, the Commission shall, in cooperation with relevant stakeholders and organisations such as metrology and benchmarking authorities, encourage, as appropriate, the development of benchmarks and measurement methodologies.
3. The levels of accuracy and the relevant accuracy metrics of high-risk AI systems shall be declared in the accompanying instructions of use.
4. High-risk AI systems shall be as resilient as possible regarding errors, faults or inconsistencies that may occur within the system or the environment in which the system operates, in particular due to their interaction with natural persons or other systems. Technical and organisational measures shall be taken in this regard. The robustness of high-risk AI systems may be achieved through technical redundancy solutions, which may include backup or fail-safe plans. High-risk AI systems that continue to learn after being placed on the market or put into service shall be developed in such a way as to eliminate or reduce as far as possible the risk of possibly biased outputs influencing input for future operations (feedback loops), and as to ensure that any such feedback loops are duly addressed with appropriate mitigation measures.

Four clauses, four jobs. Paragraph 1 sets the target: appropriate accuracy, appropriate robustness, and consistency over time. Paragraph 2 tells you who is supposed to define “appropriate” in general terms, and it is not you: the Commission. Paragraph 3 tells you what you owe the deployer regardless of what paragraph 2 produces: a declared accuracy level and the metrics behind it, in the instructions of use. Paragraph 4 is about the system’s behaviour under stress: errors, faults, inconsistent inputs, and for systems that keep learning after deployment, the specific hazard of feedback loops.

Paragraph 5, on cybersecurity against unauthorised third parties, belongs to the next module. This one is about the system failing on its own or under ordinary operating conditions, not about someone attacking it deliberately.

The obligations bind high-risk systems in Annex III from 2 December 2027, the date set by the 2026 Digital Omnibus. Of the three cast systems this course follows, only HireSift, Arbeta GmbH's CV screener, sits in that category. But the discipline in this chapter is not gated behind a legal deadline. BriskDesk and VibraSense will benefit from declared metrics and robustness tests whether or not a regulator ever asks for them, because the alternative is shipping a system nobody has measured.

Appropriate level, no official ruler yet

Paragraph 2 is doing something specific: it hands the technical definition of "appropriate" to a future benchmark, not to today's provider judgment alone, and not to today's regulator either. Read it again: the Commission "shall… encourage, as appropriate, the development of benchmarks and measurement methodologies." Encourage. Not publish, not mandate, not certify.

> AS OF MID-2026
>
> No benchmark or measurement methodology for Article 15 has been adopted, and zero AI Act harmonised standards of any kind are cited in the Official Journal. The standardisation work that would cover this article, an AI trustworthiness framework plus European adoptions of the ISO/IEC 24029 robustness series, is still moving through the CEN-CENELEC pipeline; the furthest-along deliverable of the whole programme, the quality-management standard prEN 18286, sits at Formal Vote. None of it carries the Article 40 presumption of conformity yet, because that presumption only attaches once a standard's reference is published in the OJ.

Put plainly: if you are building a high-risk system today, nobody is going to hand you a target number. There is no official F1 threshold for a CV screener, no official false-alarm

rate for a maintenance model, no ISO test suite you run and pass. This is not a gap in your knowledge. It is a gap in the standard, and it is one every provider building high-risk AI in 2026 shares.

This does not mean "appropriate" means arbitrary. It means the burden of definition sits with you as the provider, and the way you discharge that burden is documentation, not intuition. Three things make a chosen metric defensible in an audit: it must be traceable to the system's intended purpose (an accuracy metric for a ranking system looks different from one for a generative chatbot), it must be measured against a test population that resembles deployment conditions (the metric on last year's applicant pool means something different from the metric on training data), and it must be declared, in writing, where the deployer can see it. Do those three things and "provider-defined" reads as engineering judgment. Skip any of them and it reads as guesswork with better formatting.

> WATCH OUT
>
> The ISO/IEC 24029 family (robustness assessment of neural networks) sometimes comes up in this conversation. Treat it as useful scaffolding for how to think about robustness testing, not as a shortcut to compliance. It is not cited in the AI Act and carries no legal presumption. Same caution applies to ISO/IEC 42001: a certified AI management system is a real asset, but it does not, on its own, satisfy Article 15.

Declaring metrics: three systems, three declarations

Paragraph 3 is the concrete deliverable: declare the accuracy levels and the metrics behind them, in the instructions of use. "Declare" here means a specific, named number (or range) attached to a specific, named metric, attached to a specific, named test population, sitting in a document the deployer actually reads before they switch the system on. It is not a marketing claim ("industry-leading accuracy") and it is not a training-set number quietly assumed to hold in production.

Different systems need different metrics, because “accuracy” as a single word hides very different engineering questions depending on what the system does. Working through the cast makes the point concrete.

HireSift (Arbeta GmbH, the Annex III ranking system) is not a single-label classifier, so a bare accuracy percentage says almost nothing useful. What HireSift needs declared is a small metric family: precision at the shortlist cutoff (of the candidates HireSift puts through, what fraction genuinely meet the job profile), recall of qualified candidates (of the candidates who would have been strong hires, what fraction survive the filter), and, carried over directly from Module 3’s bias work, the subgroup true-positive-rate gap across the protected characteristics Arbeta examined. A ranking system that reports overall precision without the subgroup breakdown has declared a number that hides the exact failure mode regulators and plaintiffs will ask about first. The declaration also has to name the test population: precision measured on a curated benchmark of “obviously qualified” CVs is a different, weaker claim than precision measured on the actual applicant pool for a mid-size logistics company, gaps included.

BriskDesk (Solvana s.r.o., the limited-risk support chatbot) is not high-risk and Article 15 does not bind it. Solvana declares metrics anyway, because a chatbot that occasionally invents an order status or a return policy is a support-cost and trust problem regardless of legal tier. The metrics that matter here are answer correctness rate against a curated question set, a grounding or faithfulness rate (does the answer actually follow from the retrieved product and help-center content, or did the model fill a gap with something plausible), and escalation precision (when BriskDesk decides a query needs a human, is that decision usually right, not just frequent). Declaring these numbers is Solvana’s own discipline, useful evidence if a client e-shop ever asks how the widget was tested, but not a legal obligation the way it is for HireSift.

VibraSense (Norrfelt Industrial AB, the predictive-maintenance model) sits outside Annex III on the current facts, but Norrfelt still documents the assessment under Article 6(4) and keeps a declared-metrics section as part of that memo, because the honest nuance is that the classification could flip if a future version is wired to derate or stop a pump. The metrics that fit a failure-prediction model are precision-recall AUC rather than plain accuracy (bearing failures are rare, so a model that always predicts

“fine” scores misleadingly well on raw accuracy), lead time (how many days of warning the alert gives before the failure would actually occur), and false-alarm rate (how often a maintenance visit gets scheduled for a bearing that was never going to fail, because sending a technician on a false alarm has a real cost). The worksheet at the end of this chapter asks you to write this section yourself.

The pattern across all three: the right metric follows the task, not a generic template, and the declaration is only useful if it names the test population alongside the number.

Testing classical ML versus testing LLM systems

Before the eval ladder, it helps to notice why this module spends so much time on evaluation architecture rather than a single test suite. HireSift’s ranking model is closer to classical ML: a fixed feature space, a held-out test set, metrics that are stable and reproducible run to run. You compute precision at k once, on a frozen set, and the number does not move unless the model or the data moves.

BriskDesk is a different animal. The generation step is nondeterministic, the input space is effectively unbounded (anyone can type anything into a chat widget), and “correct” is often a judgment call rather than a string match. Two people reading the same BriskDesk answer might disagree about whether it was helpful enough. That is exactly the gap that has produced, over the last few years, a maturity ladder for evaluating generative systems: start where classical testing still works, and add machinery only where it stops working.

The eval maturity ladder

The four rungs below mirror the four notebooks in the `testing-tutorial` repo (`01_llm_testing_manual_to_automated.ipynb` through `04_evaluation_pipeline.ipynb`), built around an IT-support agent that stands in for BriskDesk’s own shape of problem: technical questions, tool calls, and subjective response quality all in one system.

Rung one: manual spot checks

The starting point for every team is someone reading outputs by hand. It catches obvious failures fast and costs nothing to set up. It does not scale past a handful of cases, it is not reproducible (the same reviewer on a different day may judge differently), and it produces no artifact an auditor can look at later. Useful for the first week of building a feature. Useless as ongoing evidence.

Rung two: automated assertions

The first real step up is `pytest`, run the way you would run it against any other code: fixed inputs, parameterized over a list of cases, one assertion per behaviour you actually care about.

```
import pytest

@pytest.mark.parametrize("question,expected_answer", [
    ("What port does HTTPS use? Answer with just the number.", "443"),
    ("What port does SSH use? Answer with just the number.", "22"),
    ("What does DNS stand for? Answer in one sentence.", "domain name system"),
])
def test_technical_knowledge(question, expected_answer, ask_agent):
    response = ask_agent(question).lower()
    assert expected_answer.lower() in response
```

One function, three test cases, three separate pass/fail rows in the report. Adding a fourth case is a one-line change, not a new function. This rung also covers structured-output validation with Pydantic: when BriskDesk's summarizer or HireSift's shortlist writer is asked to return JSON, a Pydantic model turns "looks like valid JSON" into an enforced, typed contract.

```
from pydantic import BaseModel, Field
from typing import Literal

class ShortlistNote(BaseModel):
    candidate_id: str
    recommendation: Literal["advance", "hold", "reject"]
    rationale: str = Field(..., min_length=20)
```

```
note = ShortlistNote.model_validate_json(llm_output)
assert note.recommendation in {"advance", "hold", "reject"}
```

If the model drifts and returns a field with the wrong type or an out-of-range value, `ShortlistNote.model_validate_json` raises before the bad output ever reaches a hiring manager. This rung is cheap, deterministic, and fast, which is exactly why it should carry as much of the test suite as the task allows. It also extends to agent behaviour, not just text: does the agent call `get_order_status` with the order ID actually present in the user's message, does it refuse to call `create_return` without an explicit confirmation step first. Those are still assertions, just against a tool call and its arguments instead of a string.

Rung three: LLM-as-judge

Assertions run out of road exactly where BriskDesk's problem starts: is this answer helpful, is the tone appropriate for a frustrated customer, is the explanation clear to someone non-technical. None of those have a keyword you can grep for. The answer the field converged on is LLM-as-judge: a second model call, given the original input, the response, and an explicit scoring rubric, returns a structured score you can assert against like any other value.

```
from pydantic import BaseModel, Field

class JudgeEvaluation(BaseModel):
    score: int = Field(..., ge=1, le=5)
    reasoning: str

def judge_response(user_input, agent_output, criteria, client, model):
    prompt = f"""Evaluate the AGENT RESPONSE against the CRITERIA below.
USER INPUT: {user_input}
AGENT RESPONSE: {agent_output}
CRITERIA: {criteria}
Return ONLY JSON: {{"score": <1-5>, "reasoning": "..."}}"""
    raw = client.responses.create(model=model, input=prompt).output_text
    return JudgeEvaluation.model_validate_json(raw)

evaluation = judge_response(
```

```
    "I've been waiting for 2 hours and nobody helped me!",
    agent_output,
    "Professionalism: does the response handle the frustrated customer empathetically?",
    client, model,
)
assert evaluation.score >= 4
```

A judge call is not free, and it is not automatically trustworthy either. The next section deals with exactly that.

Rung four: the evaluation pipeline

The top rung is what turns rungs two and three from a folder of scripts into a single, repeatable machine: a reusable evaluator class that runs a mixed dataset of assertion cases, tool-call cases, and judge cases through one interface, and emits one report per run.

```
class AgentEvaluator:
    def __init__(self, agent_function):
        self.agent_function = agent_function
        self.results = []

    def evaluate_all(self, dataset):
        for case in dataset:
            output = self.agent_function(case["input"])
            result = self._route(case, output)   # assertion | tool_call | llm_judge
            self.results.append(result)
        return self.results

    def generate_report(self):
        total = len(self.results)
        passed = sum(r.passed for r in self.results)
        return {"total": total, "passed": passed, "pass_rate": passed / total * 100}
```

This is the `AgentEvaluator` pattern from `04_evaluation_pipeline.ipynb`, and it is the piece that matters most for Article 15(1)'s "consistently throughout the lifecycle" clause. Once evaluation is a class you call, not a notebook you re-read, it becomes something you can run in CI on every release, diff against the previous run's pass rate,

and archive as a dated report. That report, mixed assertions plus tool checks plus judge scores plus a timestamp, is the shape of evidence Article 15(3) is actually asking for: not a one-time benchmark run, but a machine that re-measures the declared metrics every time the system changes.

Judge reliability: who judges the judge

An LLM judge is itself a model making a judgment call, which means it inherits every failure mode a judged model has, plus a few of its own. Before a judge's scores count as evidence for anything, three reliability properties need to be measured, not assumed.

Agreement with human labels. Take a sample of cases the judge scored, have a human independently score the same cases against the same rubric, and measure the agreement rate. A judge that agrees with human raters 60% of the time functions as a coin flip with extra steps, not a quality gate. Teams that skip this step are trusting a black box to grade another black box.

Position bias. When a judge compares two candidate outputs (for example, before-and-after a prompt change), its verdict can shift depending on which response is presented first, independent of actual quality. The standard check is to run the comparison twice with the order swapped and see whether the verdict flips. If it does, the judge is responding to position, not content, and pairwise comparisons need either averaging over both orders or a redesigned single-response scoring rubric instead.

Self-preference. A judge built on the same model family as the system being judged tends to score that family's outputs more favourably, all else equal. This matters directly for the eval-ladder pipeline: if BriskDesk's chatbot and its judge are both calls to the same hosted model, a silent thumb sits on the scale. Using a different model as judge, or periodically cross-checking with a second judge model, is the practical mitigation.

None of this means abandon LLM-as-judge. It means the judge's reliability is itself a metric, measured on a cadence, documented alongside the declared accuracy metrics from Article 15(3). "Our judge agrees with human raters at 84% on a 50-case calibration set, re-measured quarterly" is a sentence an auditor can act on. "We used an LLM to check quality" is not.

Robustness under Article 15(4): the threat table

Paragraph 1 asked for robustness. Paragraph 4 says what that means operationally: resilience to errors, faults, and inconsistencies, achieved where useful through technical redundancy, and, for systems that keep learning, explicit control of feedback loops. The table below expands each named threat into a concrete test and the artifact that test produces.

THREAT	WHAT IT LOOKS LIKE	TEST	EVIDENCE ARTIFACT
Malformed or adversarial-benign input	A CV with corrupted encoding, a chat message with broken markdown, a sensor reading outside the sane range	Fuzz testing and property-based tests over the input schema	Fuzz-test report, crash/exception count
Out-of-distribution input	HireSift receives a CV in a format or language never seen in training; BriskDesk gets a question about a product line it has no catalog entry for	A curated OOD input set, checked for graceful refusal or fallback rather than a confident wrong answer	OOD test suite pass rate
Noise and perturbation	Small paraphrases of a support question; sensor jitter on VibraSense's vibration feed	Run the eval set clean, then run perturbed copies of the same set, and compare pass rates	Clean-vs-perturbed delta report
Distribution shift and drift	VibraSense's pump fleet ages, gets serviced, or gets replaced with different hardware; HireSift's applicant pool shifts after a hiring-market change	Production monitoring metrics on live input and output distributions, with alert thresholds	Drift dashboard, alert log

THREAT	WHAT IT LOOKS LIKE	TEST	EVIDENCE ARTIFACT
Upstream model swap or API change	BriskDesk's hosted LLM provider silently updates the model behind an API version tag	A pinned regression suite re-run against the new model version before it is adopted	Regression-suite diff report
Feedback loops (Article 15(4) explicit)	HireSift's screening decisions become part of next year's "successful hire" training data, quietly amplifying whatever bias shaped this year's decisions	Outcome monitoring on downstream decisions plus retraining guards that block feeding recent model outputs back in as unexamined ground truth	Feedback-loop audit log, retraining data-lineage record

The upstream model swap row deserves a second look, because it is the one developers most often miss. BriskDesk does not train its own model; it calls a frontier model through a hosted API. That means Solvana does not control when the underlying weights change, and a provider-side update can shift BriskDesk's behaviour with zero code change on Solvana's side. The only defence is the same one classical software engineering already has for a third-party dependency bump: pin the version where the API allows it, and when a new version becomes unavoidable, run the full pinned regression suite against it before treating the new version as adopted. If the regression suite catches a drop in grounding rate or a change in refusal behaviour, that is the signal to hold the upgrade, not ship it silently.

HireSift's feedback loop row is the other one worth sitting with, because it is the row Article 15(4) calls out by name. HireSift's ranking model was trained on ten years of historical hiring outcomes (Module 3 covered how that history carries its own bias). Every year HireSift operates, this year's shortlisting decisions become part of next year's outcome data, unless Arbeta deliberately breaks that cycle. A candidate HireSift never shortlisted can never generate a "successful hire" label, so the model's own past decisions quietly become tomorrow's ground truth, amplifying whatever it already got wrong. The mitigation runs through a governance rule rather than a single test:

retraining pipelines flag or exclude recent model-influenced outcomes from unexamined reuse as training labels, and someone reviews that exclusion policy on a schedule, not once at launch.

Consistently throughout the lifecycle: the monitoring bridge

Everything in this chapter so far describes what happens at release: a declared metric, a test suite, a robustness table, all checked before the system ships. Article 15(1) does not stop there. “Perform consistently… throughout their lifecycle” makes release-time testing necessary but not sufficient. A model that passed every check on launch day can still drift, decay, or degrade six months later, quietly, with nobody watching.

This is the seam where Article 15 hands off to Article 72. Article 72 requires providers to establish and document a post-market monitoring system, proportionate to the system’s nature and risk, that actively and systematically collects, documents, and analyses relevant performance data throughout the system’s time on the market. Read together, Article 15 tells you what to measure and declare, and Article 72 tells you to keep measuring it after the conformity assessment is filed away.

In practice this closes into a loop: a drift alarm fires on a monitoring dashboard (VibraSense’s sensor-input distribution has shifted because the fleet composition changed), which triggers a re-run of the declared-metrics suite against current data, which produces either an unchanged declaration (the system still meets what was declared) or an updated one (the declared metrics need revising, and the instructions of use need to say so). That loop, alarm, re-run, updated declaration, is the concrete shape of “consistently throughout the lifecycle.” Module 9 builds the logging and CI machinery that runs this loop automatically; this module builds the metrics and tests the loop re-runs.

Worksheet: declare VibraSense’s metrics

Norrfelt Industrial AB keeps a documentation memo justifying why VibraSense sits outside Annex III today (Module 2 covered the reasoning: it schedules maintenance, it does not actuate anything). Even without a legal Article 15(3) duty, the memo includes a

declared-metrics section, because “not high-risk today” is a conclusion that could flip, and because a maintenance model nobody has measured is a bad idea regardless of legal tier.

Using what you now know about VibraSense (a gradient-boosted classifier over vibration and temperature time series, scored nightly in batch, outputting a failure-risk score that schedules service visits), write the declared-metrics section for Norrfelt’s memo. Cover:

1. Which metric or metrics you would declare, and why a plain accuracy percentage would be misleading for this task.
2. What test population the metric should be measured against, and why last year’s fleet data might not be enough on its own.
3. One robustness test from the threat table above that matters most for VibraSense specifically, and what artifact it should produce.

Write your answer before reading the appendix. A sample answer follows at the end of this chapter, but the value of the exercise is in the choices you make before you see someone else’s.

Appendix: sample answer

Metrics declared. VibraSense declares three numbers, not one: precision-recall AUC rather than raw accuracy, because bearing failures are rare events and a model that predicts “no failure” for every pump would score misleadingly high on plain accuracy while catching nothing. Alongside PR-AUC, VibraSense declares median lead time (the number of days between the alert and the actual failure, since a technically correct alert that arrives the day of failure is operationally useless) and false-alarm rate (the fraction of scheduled service visits that found nothing wrong, since each false alarm is a real cost in technician time and customer goodwill).

Test population. These metrics are measured against a held-out set of pump-years drawn from Norrfelt’s actual serviced fleet, not a synthetic or vendor benchmark, and the memo names the date range and fleet composition the test set covers. Last year’s

fleet data alone is not enough, because fleet composition changes: pumps get replaced with newer hardware, sensors get recalibrated or swapped for a different model, and a metric measured only on last year's sensors quietly stops describing this year's population. The memo commits to refreshing the test population at each hardware-generation change, not on a fixed calendar alone.

Robustness test. The threat that matters most for VibraSense is distribution shift and drift, specifically sensor aging: a vibration sensor's baseline readings shift subtly as the sensor itself ages, which can look like early bearing wear to a model trained on newer-sensor data, or conversely mask real wear behind sensor noise the model was never trained on. The test is a monitoring metric comparing the live input distribution against the training distribution on a rolling window, with an alert threshold that triggers a re-run of the declared PR-AUC and false-alarm-rate suite against recent data. The artifact is a drift dashboard plus a dated log of every alert and the metric re-run it triggered, which is exactly the artifact Article 72's post-market monitoring system asks a provider to keep, whether or not VibraSense ever crosses into Annex III.

CHAPTER 5

Adversarial Robustness and Cybersecurity, Article 15 Part Two

Module 4 asked whether your system is accurate. This module asks whether someone can make it wrong on purpose. Article 15(5) binds high-risk systems from 2 December 2027, the same fixed date as the rest of Chapter III Section 2. But the attacks it describes do not wait for a compliance deadline. If your system takes user input, retrieves documents, or calls a tool, someone can try to break it today, whatever risk lane it sits in. This chapter treats security as an engineering discipline you build now and prove with tests, not a control you retrofit before an audit.

Article 15(5): the law names the attacks

Here is the article, in full, from Chapter III Section 2:

> THE ACT SAYS · ARTICLE 15(5)
>
> High-risk AI systems shall be resilient against attempts by unauthorised third parties to alter their use, outputs or performance by exploiting system vulnerabilities.
>
> The technical solutions aiming to ensure the cybersecurity of high-risk AI systems shall be appropriate to the relevant circumstances and the risks.
>
> The technical solutions to address AI specific vulnerabilities shall include, where appropriate, measures to prevent, detect, respond to, resolve and control for attacks trying to manipulate the training data set (data poisoning), or pre-trained components used in training (model poisoning), inputs designed to cause the AI model to make a mistake (adversarial examples or model evasion), confidentiality attacks or model flaws.

Read it slowly, because it is doing three distinct jobs.

The first paragraph sets the standard: resilience against unauthorised third parties who exploit vulnerabilities to alter use, outputs, or performance. That is a general resilience duty, deliberately technology-neutral.

The second paragraph is the proportionality clause: solutions must be "appropriate to the relevant circumstances and the risks." This is not a loophole. It is an instruction to size your defenses to your actual attack surface and stakes, not to build every conceivable control for every system regardless of exposure. A read-only FAQ bot and a system with write access to a payment API do not need the same defense budget, and the law says so explicitly.

The third paragraph is where the Act gets specific, and this is the part worth memorizing. It names four attack classes by their engineering names, not by vague reference to "cybersecurity":

- **Data poisoning**: attacks trying to manipulate the training data set.
- **Model poisoning**: attacks trying to manipulate pre-trained components used in training.
- **Adversarial examples or model evasion**: inputs designed to cause the AI model to make a mistake.
- **Confidentiality attacks or model flaws**: a catch-all closing the list.

And it names the five required capabilities as verbs: prevent, detect, respond to, resolve, and control. Notice that four of the five are not about stopping the attack before it happens. Detect, respond, resolve, and control assume an attack will get through sometimes, and ask what you do next. As engineering, this reads as: controls, plus tests that exercise those controls, plus monitoring that would catch a live attack, plus a runbook for when one succeeds. Controls alone satisfy none of the five verbs on their own; a firewall with no logging detects nothing, and a logging system with no response plan resolves nothing.

Here is the gap worth naming honestly. The list does not say "prompt injection." The Act's drafters were writing against a threat model closer to classical adversarial machine learning, perturbed pixels that flip an image classifier, poisoned labels that skew a

scoring model, the kind of attack research that predates the current wave of tool-calling LLM agents. Prompt injection, the attack where an LLM follows instructions embedded in its input rather than the instructions its operator intended, is not named anywhere in Article 15(5).

That does not put it outside the article. It falls under "inputs designed to cause the AI model to make a mistake": a prompt injection is exactly an input engineered to make the model do something its designer did not intend. It also plausibly falls under the closing catch-all, "model flaws." The legal text supplies the obligation; it does not supply the vocabulary for the attack surface that showed up after the drafting closed. That vocabulary comes from industry practice, and the reference most engineering teams reach for is the OWASP LLM Top 10 (LLM01 prompt injection, LLM02 sensitive information disclosure, LLM04 data and model poisoning, and more). Treat OWASP's list as the map that overlays the Act's four legal classes with the specific attacks an LLM system actually faces. Your job as the engineer is the gap-filling: the law tells you the outcome required (resilient, tested, documented), the industry frameworks tell you which attacks to test for.

Threat modeling for LLM systems

Before you can defend a system you need a threat model, and a threat model for an LLM-based system asks three questions.

Assets. What does an attacker want? Not "the AI model" in the abstract, but the specific things of value: a tool that moves money or issues refunds, a database of personal records, the system prompt encoding business logic you would rather a competitor not see, the trust a user places in the system's output.

Entry points. Where does untrusted content reach the model? The obvious one is the chat box. The less obvious ones are everything the system retrieves or is told to read on the user's behalf: documents from a retrieval index, emails, product reviews, the contents of a tool's response, a file a user uploads.

Trust boundaries. Where does the trust level change as data moves through the system? A boundary crossing is where you need a check, because everything on the far

side of it should be treated as adversarial until proven otherwise.

Walk this through BriskDesk in full, because it is the running example for exactly this reason: it is a RAG system with tool access, which is the shape most LLM applications in production actually have.

BriskDesk is Solvana's customer-support chatbot. It answers product questions using retrieval over each client's product catalog and help-center articles, and it can call two tools against the client's order API: `get_order_status` and `create_return`.

Assets: the `create_return` tool (it changes state and costs the client money), the order-lookup capability (it exposes customer order data), the RAG index itself (poisoning it poisons every future answer that retrieves from it), and the system prompt encoding which questions the bot should and should not answer.

Entry points: the chat widget, where any site visitor types text directly (this is the direct-injection surface); the RAG index, which ingests the client's product catalog and help-center articles, and in many real deployments also ingests or is adjacent to user-generated content like product reviews (this is an indirect-injection surface, because the attacker never talks to BriskDesk, they talk to the e-shop's review form); and the order API's responses, which the model reads as tool output and could, in principle, contain attacker-influenced fields if the attacker has any way to write into an order record before BriskDesk reads it back.

Trust boundaries: the boundary between "text a real customer typed" and "the system prompt" is boundary one, and every LLM chatbot has it. The boundary between "content retrieved from the index" and "instructions the operator wrote" is boundary two, and it is the one BriskDesk's designers are most likely to overlook, because the index looks like the bot's own data, not like user input. The boundary between "the model decided to call a tool" and "the tool actually executes" is boundary three, and it is the one that turns a successful injection into an actual loss.

Now the scenario the slides and the M05 exercise both build on. An attacker posts a product review on the client's e-shop. The review text looks like an ordinary complaint, but embedded in it are instructions aimed at whatever system might read the review

later, not at a human shopper. The e-shop's normal content pipeline embeds the review and adds it to BriskDesk's vector store, because that is what the pipeline does with all reviews. Some time later, an ordinary customer asks BriskDesk an ordinary product question. Retrieval brings back the poisoned review as relevant context, because vector similarity does not know or care that the review was written for the model rather than for a person. The model reads its context window, sees the review's embedded instructions sitting right next to the legitimate product information, and, without a control to stop it, follows them: calling `create_return` for the customer who only asked a question.

The lesson generalizes past BriskDesk: the model reads instructions wherever they sit in its context, regardless of whether you, the engineer, intended that content as data or as instructions. Everything retrieved, and everything a tool returns, is a place an attacker can write to if they can influence that source at all.

Prompt injection in depth

Split the attack into three shapes, because each needs a different defense.

Direct injection. The attacker is the user. They type text at the chat box aimed at overriding the system prompt or extracting behavior the operator did not intend. This is the shape most people picture first, and it is the one system-prompt hardening and refusal training address most directly.

Indirect injection. The attacker never talks to the system. They plant instructions in content the system is designed to read as data: a RAG document, an email the system summarizes, a product review, a web page a browsing tool fetches. BriskDesk's poisoned review is this shape. Indirect injection is harder to defend because the content genuinely is data from the system's point of view; the attack works precisely because the model does not reliably distinguish "text to answer questions about" from "text to obey."

Tool abuse. A variant of indirect injection specific to agentic systems: the payload arrives through a tool's return value rather than through a document. If `get_order_status` returns a free-text notes field that a customer (or an attacker

posing as one) can write into an order record, and the model reads that field as part of deciding what to do next, the tool's own output becomes an injection vector.

> AS OF MID-2026
>
> None of these are solved problems. What exists is defense in depth: layered mitigations, each independently imperfect, that together shrink the attack surface and, just as important, generate the evidence Article 15(5)'s "detect" and "respond" verbs ask for.

Input source tagging. Before you can defend against instructions arriving disguised as data, you need to know, in code, which context came from where. Tag every block of content entering the prompt with its provenance and trust level, and keep that tag attached through the whole pipeline rather than flattening everything into one string.

```
from dataclasses import dataclass

@dataclass
class ContextBlock:
    source: str          # "user_input" | "rag_retrieval" | "tool_result"
    trust_level: str     # "untrusted" | "semi_trusted" | "system"
    content: str

def build_prompt(blocks: list[ContextBlock]) -> str:
    sections = []
    for b in blocks:
        # Untrusted content is labelled explicitly, never blended
        # silently into the instruction stream.
        sections.append(f"[SOURCE: {b.source} | TRUST: {b.trust_level}]\n{b.content}")
    return "\n\n".join(sections)
```

This does not stop a determined injection by itself. What it does is make the trust distinction visible to any downstream logic, including a guardrail model, and it is a prerequisite for every other control in this list. You cannot check what you have not labelled.

Least-privilege tool design. The single most effective control for BriskDesk's scenario is architectural, not a prompt technique: the tool that looks up an order should not be the same tool, or even the same permission scope, as the tool that changes one.

```
class OrderLookupTool:
    """Read-only. Cannot be escalated to a write action."""
    def get_order_status(self, order_id: str, session_customer_id: str) -> dict:
        order = order_store.fetch(order_id)
        if order.customer_id != session_customer_id:
            raise PermissionError("order does not belong to this session")
        return order.to_public_dict()

class RefundTool:
    """Write. Requires an explicit, separately-authorised confirmation
    step; the model's own output is never sufficient authorisation."""
    def create_return(self, order_id: str, session_customer_id: str,
                      human_confirmed: bool) -> dict:
        if not human_confirmed:
            raise PermissionError("create_return requires confirmation")
        ...
```

The model can be talked into wanting to call `create_return`. It cannot be talked into a permission it was never granted, because the permission lives in code, not in the conversation. This is the same principle as least-privilege in any other system: the blast radius of a successful injection is capped by what the compromised component was allowed to do in the first place, not by how well it resisted persuasion.

Output and pre-execution checks. Before a consequential tool call actually executes, run an independent check that does not trust the model's stated reasoning.

```
def pre_execution_check(action: str, args: dict, ctx: RequestContext) -> bool:
    if action == "create_return":
        # Was this action requested by the authenticated session's own
        # customer, not merely mentioned somewhere in retrieved context?
        if args["order_id"] not in ctx.session_order_ids:
            log_security_event("blocked_return_out_of_session", ctx, args)
            return False
        if not ctx.explicit_user_confirmation_seen:
            log_security_event("blocked_return_no_confirmation", ctx, args)
```

```
            return False
    return True
```

> **WATCH OUT**
>
> This is a guardrail, not a proof. A sufficiently creative attack path can still find a gap. The point of stacking source tagging, least-privilege tools, and pre-execution checks is that an attacker now has to defeat three independent layers rather than one, and every blocked attempt is a logged event, which is the evidence Article 15(5) actually asks for.

Poisoning and supply chain

Article 15(5) names two poisoning classes separately, and they need different controls.

Training data poisoning targets the data set your model learns from. HireSift is the clean example: its ranking model retrains on ten years of historical hiring outcomes submitted by client companies. If a client, or anyone with write access to that outcome store, can inject fabricated outcome labels, they can shift how the model ranks future candidates, and because the model retrains periodically, the effect compounds. The control is not a model-side fix; it is a pipeline-side one. Decide, in writing, who is authorized to write to the training data store. Track provenance on every record: which client submitted it, when, through what channel. Watch for anomalies in the label distribution before a batch enters training, and quarantine anything that looks statistically off rather than merging it automatically. This is the write-access-control question the module's quiz comes back to, and it belongs at the ingestion point, before the data ever reaches a training run, not at the output stage where a hiring manager sees a ranked shortlist.

Model poisoning targets pre-trained components rather than your own data: a base model, an embeddings model, a fine-tuned checkpoint pulled from a public hub. This is a supply-chain problem with a software-engineering analogue you already know. Treat a downloaded checkpoint the way you would treat a third-party dependency: verify its provenance and hash, pin the version you tested against, read its model card for known

issues, and do not silently auto-update to “latest” in a production pipeline. The AI-specific twist is that a poisoned checkpoint is much harder to diff than a poisoned package, because there is no source code to review, only weights. That is exactly why provenance and pinning matter more here, not less.

Eval-set contamination is not named in Article 15(5)’s text, but it belongs in the same family and it connects directly back to Module 4’s accuracy work. If your benchmark or evaluation set leaks into a model’s training data, directly or through a vendor’s pretraining corpus, the declared accuracy figure Article 15(3) requires you to publish is no longer measuring what it claims to measure. The control is eval hygiene: hold out evaluation data you control, seed it with canary strings you can search for later, and periodically check whether a model’s performance on your eval set is suspiciously close to memorization rather than generalization. This is not a one-time check; every new model version deserves a fresh contamination pass before you trust its eval numbers.

Confidentiality attacks, at concept level

Article 15(5)’s fourth named class is confidentiality attacks. Two patterns matter most for LLM systems, described here at the level you need to defend against them, not as a how-to.

System-prompt extraction. An attacker tries to get the model to reveal its own instructions, its tool definitions, or internal business rules through crafted questioning. The defensive posture that actually works is not “write a system prompt so clever it never leaks.” Treat your system prompt the way you would treat client-side JavaScript: assume it will eventually be read by someone determined enough, and design your security boundary somewhere else entirely. The real boundary for BriskDesk is server-side authorization on what `create_return` will actually execute, not whether the model can be talked into printing its own instructions. A leaked system prompt is embarrassing. A leaked system prompt that also grants an actual capability is the failure worth engineering against.

Membership inference. An attacker who can query a model repeatedly, and observe its outputs or confidence closely enough, can sometimes infer whether a specific record was present in its training data. This matters most where the training data is itself sensitive, which is exactly HireSift's situation: candidate CVs are the crown-jewel asset, and a summarizer or ranking model trained on real candidate histories carries some risk that a sufficiently probing query pattern could reveal whether a particular person's data was used. The mitigation is less a model technique and more an access-and-logging discipline: rate limits and anomaly detection on query patterns against sensitive models, and access logging thorough enough that Module 6's audit trail can show who queried what, and how often, after the fact.

The warden case study

Threat modeling and layered defenses are qualitative. Article 15(5)'s "detect" verb asks for something you actually measured. The warden study is an example of what that measurement looks like, and it is worth walking through honestly, limits included.

The question: does adding an LLM-as-judge layer, a second model that reviews the first model's response before it reaches the user, actually reduce successful jailbreaks against a target system, and does the answer depend on how you build the judge?

The setup: twenty public jailbreak patterns (referenced only by category label, for example the zetalib and synth families used in the underlying study, never by their actual attack text), run against three open-weight target models, crossed against four rule sets and four judge designs and two placements for where the judge sits in the pipeline. Every trial logged in full: the verdict, the judge's reasoning, token counts, and cost.

The headline result: with no judge layer in place, the baseline attack success rate across the twenty patterns and three targets averaged roughly sixteen percent. That average hides the study's other finding: the three targets' own baselines ranged from five percent to about twenty-four percent, a nearly five-fold spread in how much resistance the open-weight models ship with before any judge is added. Adding a judge

layer dropped that substantially, down to around one percent for the best-performing judge design.

The honesty, which matters as much as the headline. That one-percent figure is an average across judged configurations, not a property of "a judge layer" as a generic claim. Results vary meaningfully by judge design and by placement: in this run the strongest configurations blocked every attack, the weakest judged configuration let roughly two percent through, and pooled across all judged trials a small number of attacks still landed (the module notebook computes the exact counts and shows which attack categories they came from). A judge layer is also not free: it adds a full model call's worth of latency to every response, and it adds token cost that scales with traffic, both of which are exactly the kind of tradeoff Article 15(5)'s proportionality clause expects you to weigh rather than treat as a mandatory checkbox.

Map this back to the article's five verbs. The trial log itself, twenty patterns times three targets times four rules times four judges times two placements, all captured, is your evidence for "detect." The judge layer's measured drop in attack success is your evidence for "prevent" and "respond." The residual attacks that still get through under every configuration are not a result to bury; they are your "resolve and control" gap, and documenting a known, measured gap is stronger evidence of a functioning security program than a document that claims no gap exists.

Security testing as a regression suite

Module 4 built an evaluation pipeline in four stages: manual factual checks, automated tests, an LLM-as-judge layer, and a CI-wired eval suite that reruns on every release. Security testing is the same pipeline aimed at a different question. Instead of "did the model answer correctly," it asks "did the model resist a known attack class."

The practical shape: every attack class you have identified in your threat model, the poisoned-review scenario, a system-prompt extraction attempt, a guardrail-bypass probe, becomes a test case with a known expected outcome (refuse, or execute only the authorized action). Every time you fix a gap, the fix becomes a permanent regression test, because a defense that worked once and was never re-checked is only a hope that it

still works, not something you can document as a defense. Wire the suite into the same CI job Module 9 builds for post-market monitoring, so a new model version, a new prompt template, or a new tool addition all re-run the full security regression suite before anything ships. The alternative, testing security once at launch and never again, is how a working control quietly stops working the day a dependency updates.

Building the security_tests.md evidence section

The artifact this module produces, whether generated from the bundled warden excerpt in the accompanying notebook or from your own live testing against your own system, has five parts, and each one answers a question an auditor will actually ask.

Method. What you tested and how: which attack classes, against which models, using which tooling (a live path with real API calls, a cached-results analysis, or both).

Scope. What was in bounds and what was out. A test suite that only covers direct injection while your system's real exposure is indirect injection through RAG is a scope gap worth stating plainly, not hiding.

Findings. The measured results: attack success rates by category, by model, by mitigation configuration, exactly the shape of table the warden analysis produces.

Mitigations. What you built in response: the source-tagging, least-privilege tool boundaries, and pre-execution checks described earlier in this chapter, each tied to the finding it addresses.

Residual risk. What still gets through, honestly stated. This is the section most compliance documents skip, and it is the section Article 15(5)'s "resolve and control" verbs are actually asking for. A number like "roughly one percent under our best configuration, higher under others" is a stronger piece of evidence than silence, because it shows the control is measured rather than assumed.

Worksheet: threat-model VibraSense's telemetry pipeline

VibraSense, Norrfelt Industrial's predictive-maintenance model, was classified out of Annex III in Module 2: it schedules maintenance visits, it does not actuate anything, and its failure does not directly endanger health or safety. Article 15(5) does not legally bind it today. Apply the threat-modeling method from this chapter to it anyway, and write your own answer before turning to the appendix.

Its pipeline: vibration and temperature sensor time series, arriving from field devices, feed a feature pipeline, which feeds a gradient-boosted classifier, which scores failure risk in a nightly batch job, which populates a dashboard the service-planning team uses to schedule visits.

Work through the three questions:

1. **Assets.** What does an attacker, or an ordinary fault, actually threaten here? What is the crown jewel: the score itself, the training history behind it, something else?
2. **Entry points.** Where does data enter this pipeline from outside your control? Is the raw sensor feed the only entry point, or are there others, including anything a human writes that could later feed back into training?
3. **Trust boundaries.** Where does the trust level change as telemetry moves from a field device toward the classifier? What would you check at that boundary if you treated sensor data the way you treat a RAG document, as content you retrieve and must not blindly trust?

Appendix: sample answer to the VibraSense worksheet

Assets. The failure-risk score's integrity is the primary asset: a wrong score either wastes a maintenance visit or, worse, delays one that was needed. The historical failure-label data used to calibrate the classifier is a second asset, since it shapes every future score. The sensor calibration parameters themselves are a smaller third asset, relevant mainly if an attacker wanted to make the readings look normal.

Entry points. The raw vibration and temperature telemetry from field devices is the main entry point, and it deserves scrutiny: a device that has been physically tampered with, or a compromised gateway between the device and the ingestion pipeline, can

inject fabricated readings that look plausible. A second, quieter entry point is any technician-authored maintenance note that ends up feeding back into the labels used for retraining; if a customer can contest or influence a maintenance record to avoid a service charge, and that record later relabels a training example, that is a slow-motion path to data poisoning even without any malicious intent to attack the model specifically. A third entry point, easy to miss, is firmware updates to the sensor hardware itself, a supply-chain vector rather than a data one.

Trust boundaries. The boundary that matters most is where raw telemetry, coming from a field device you do not fully control, crosses into the feature pipeline you do control. That crossing deserves the same posture as a RAG document crossing into a prompt: range checks, anomaly detection against known-plausible sensor bands, and a quarantine path for readings that look physically implausible, before that data ever reaches the classifier or a future training run.

The poisoning framing, applied honestly. If historical failure labels used to train VibraSense's classifier are gameable through the maintenance-note feedback loop described above, that is data poisoning by the Act's own definition, even on a system the Act does not currently classify as high-risk. This is the same feedback-loop risk Article 15(4) raises for the accuracy side of this article, and it is worth engineering against regardless of legal status, because the failure mode (a slowly biased maintenance schedule) is real whether or not a regulation requires you to prove you addressed it.

The verdict, stated without hedging. VibraSense is not high-risk under Annex III today, and the threat model above does not create a legal Article 15(5) obligation. It remains good engineering practice under the accuracy and robustness duties Module 4 covered. The nuance worth remembering from Module 2 stands here too: the moment a future version of VibraSense is wired to derate or stop a pump directly rather than only alert a human, its classification reopens, and this exact threat model stops being a good habit and becomes the Article 15(5) obligation this chapter has been building toward the whole way through.

CHAPTER 6

Logging That Satisfies Auditors: Article 12 as an Engineering Spec

Module 4 asked whether the system works and keeps working. This module asks a narrower, more mechanical question: when something goes wrong, or a regulator asks, can you prove what happened? Article 12 of the AI Act is the article that makes the answer to that question a design requirement rather than a hope. It does not ask you to write more documentation. It asks you to build a system that writes its own evidence as it runs.

That distinction matters for how you read this chapter. Everything else in Chapter III Section 2, accuracy metrics, robustness tests, oversight measures, produces a claim about the system. Article 12 produces the record that lets someone check the claim later, after the system has been running for months, possibly after the people who built it have moved on. Logging is not one obligation among several. It is the evidence substrate the other obligations stand on: an incident report under Article 73 is built from logs, post-market monitoring under Article 72 consumes logs, and Module 7's oversight measurement replays logs to show a human actually looked. Get the schema wrong here and every downstream duty inherits the gap.

Reading Article 12 clause by clause

Here is the text, unmodified, from Chapter III Section 2 of the enacted Act:

> THE ACT SAYS · ARTICLE 12
>
> 1. High-risk AI systems shall technically allow for the automatic recording of events (logs) over the lifetime of the system.
> 2. In order to ensure a level of traceability of the functioning of a high-risk AI system that is appropriate to the intended purpose of the system, logging capabilities shall enable the recording of events relevant for: (a) identifying situations that may result in the high-risk AI system presenting a risk within the meaning of Article 79(1) or in a substantial modification; (b) facilitating the post-market monitoring referred to in Article 72; and (c) monitoring the operation of high-risk AI systems referred to in Article 26(5).
> 3. For high-risk AI systems referred to in point 1(a) of Annex III, the logging capabilities shall provide, at a minimum: (a) recording of the period of each use of the system (start date and time and end date and time of each use); (b) the reference database against which input data has been checked by the system; (c) the input data for which the search has led to a match; (d) the identification of the natural persons involved in the verification of the results, as referred to in Article 14(5).

Three paragraphs, three jobs. Paragraph 1 sets the baseline capability. Paragraph 2 tells you why that capability exists, which turns out to be the more useful clause for schema design. Paragraph 3 hands you, for one narrow category of system, an actual worked example of what a compliant log looks like. Read in order, they move from “you must be able to do this” to “here is why” to “here is what it looks like when someone wrote it down for you.”

12(1): a capability, not a policy

Paragraph 1 says the system “shall technically allow for the automatic recording of events (logs) over the lifetime of the system.” Two words carry the weight of the whole clause, and both are easy to read past on a first pass.

The first is “technically allow.” This is not a policy statement. A company handbook that says “employees must ensure the system logs important events” does not satisfy Article 12. The Act asks for a capability built into the system itself: the software has to be able to write a log entry when an event happens, without a human remembering to flip a switch. That is a design decision made at build time, not a compliance decision made at audit time. If logging was never wired into the system’s architecture, no amount of after-the-fact process fixes the gap, because the events that should have been recorded are simply gone.

The second is “lifetime.” Not launch week, not the pilot period, not the twelve months covered by the conformity assessment. The whole time the system is in use. A logging setup that works in the demo and silently stops after a deployment migration, a version bump, or a database cleanup script that nobody flagged as touching the log store, is not “over the lifetime” logging. This is the clause that should make an engineer ask “does our logging survive a redeploy, a schema migration, a change of hosting provider” the same way they would ask it about backups.

12(2): why you log, which is also how you design the schema

Paragraph 2 names three purposes, and each purpose implies something concrete about what an event needs to contain.

(a) Identifying risk situations and substantial modifications. If a log is going to help someone recognize that a system now presents a risk under Article 79(1), or that it has been substantially modified since its conformity assessment, the log has to capture system identity and version alongside the event. An event with no version field cannot tell an investigator whether the failure happened on the version that was assessed or a later one nobody re-assessed.

(b) Facilitating post-market monitoring (Article 72). Article 72 asks providers to actively collect and analyse performance data throughout the system’s time on the market. A log that only records failures, and never routine operation, gives post-market monitoring nothing to compute a baseline against. The schema has to record normal events, not just exceptional ones, or there is no “before” to compare the “after” to.

(c) Monitoring operation for deployers (Article 26(5)). Article 26(5) puts an operational-monitoring duty on the deployer, the organisation actually running the system day to day. That duty only works if the deployer's own copy of the logs, the one under their control, contains enough to monitor with: not necessarily the model's internals, but which decisions were made, when, and under what oversight.

Notice what these three purposes have in common: none of them is satisfied by a log that only records that "something happened." Each one needs the log to answer a specific downstream question, which is the real argument for designing a schema deliberately rather than logging whatever happens to be convenient to print.

12(3): the Act's own worked example

Paragraph 3 binds only one category of system: those in point 1(a) of Annex III, which covers biometric identification systems. Read narrowly, it is a niche obligation. Read as a design template, it is the single most useful sentence in the whole article, because it is the only place in Article 12 where the Act stops describing purposes in the abstract and actually lists fields.

The four things it names:

1. **Period of each use** (start date and time, end date and time). WHEN.
2. **The reference database checked.** AGAINST WHAT.
3. **The input data that led to a match.** WHAT.
4. **The identities of the humans who verified the result**, per Article 14(5). VERIFIED BY WHOM.

WHO, WHAT, AGAINST WHAT, VERIFIED BY WHOM, WHEN. That is a complete evidentiary record for a decision: it tells you when the system acted, on what basis, against what reference, and which human signed off. This binds only remote-biometric identification. But nothing stops a CV screener, a support chatbot, or a maintenance model from asking the same five questions of its own events. A schema that can answer "when did this run, on what input, against what reference or model, verified by whom" is a schema that has done the hard design work Article 12(3) did for one narrow case,

applied to a system that has no such narrow obligation at all. That is the reading this chapter takes forward.

An event taxonomy for an AI system

Before writing a log schema, you need to decide what counts as an event. The course's governance toolkit settles this with six event types, and the taxonomy maps directly onto the oversight verbs Article 14 names (understand, monitor, interpret, override, stop), which Module 7 covers in depth.

EVENT TYPE	WHAT IT CAPTURES	HIRESIFT EXAMPLE	BRISKDESK EXAMPLE
`tool_call`	The system invoked a function or external action	Fetching a candidate's parsed CV record from storage	`get_order_status` or `create_return` firing against the client's order API
`decision`	The system produced a scored, ranked, or filtered output	A candidate scored and ranked against a job profile	The RAG pipeline selecting which catalog passage to answer from
`review`	A human looked at an output	A recruiter opened a shortlisted candidate's summary	A support lead spot-checks a batch of chatbot transcripts
`override`	A human reversed or changed the system's output	A recruiter promotes a candidate HireSift filtered out	An agent manually corrects a wrong order status the bot gave
`stop`	The system was halted, mid-session or entirely	A hiring manager pauses a screening run pending a bias review	A blocked tool call from Module 5's pre-execution checks halts a suspicious return request

EVENT TYPE	WHAT IT CAPTURES	HIRESIFT EXAMPLE	BRISKDESK EXAMPLE
`system change`	Model version, prompt, or configuration changed	HireSift's ranking model is retrained on a new outcome batch	Solvana swaps the hosted model behind BriskDesk's API

Six types, and every one of them answers a piece of the Article 12(2) purposes: `tool_call` and `decision` build the operational trail Article 26(5) needs; `override` and `review` are the human-oversight evidence Module 7 will measure; `stop` and `system change` are exactly the events that let someone identify a risk situation or a substantial modification under 12(2)(a).

The OversightEvent record, field by field

The course's vendored governance code (`code/governance/governance/oversight_event.py`) turns this taxonomy into a concrete, runnable schema. It is a frozen dataclass with seventeen fields:

```
@dataclasses.dataclass(frozen=True)
class OversightEvent:
    timestamp: float
    system_id: str
    version: str
    session_id: str
    user_subject_id: str
    event_type: str             # tool_call | decision | override | stop | review
    tool_name: str = ""
    tool_args_hash: str = ""    # SHA-256 of canonical-JSON args; never the args themsel
    model_id: str = ""
    confidence: float | None = None
    risk_flags: tuple[str, ...] = ()
    overseer_subject_id: str = ""
    override_reason: str = ""
    fria_ref: str = ""
    causal_chain_ref: str = ""
```

```
    prev_hash: str = ""         # SHA-256 of previous event; "" for the genesis event
    self_hash: str = ""         # SHA-256 of this event's canonical form (filled by appe
```

Frozen means an `OversightEvent` cannot be mutated after construction, which is itself a small piece of tamper-resistance: nothing in the codebase can quietly edit a field on an event object that has already been created. Two fields deserve a closer look, because they are where the schema makes a deliberate trade-off rather than an obvious one.

Why `tool_args_hash`, not `tool_args`. The dataclass never stores the actual arguments a tool call was made with. It stores `hash_args(args)`, a SHA-256 digest computed over the canonical JSON form of the arguments. This is the minimization choice discussed later in this chapter, but the mechanical reason belongs here: a hash lets you prove, later, that a specific set of arguments was used (recompute the hash from a suspected value and compare), without the log itself becoming a second copy of whatever personal or sensitive data passed through the call. If HireSift's `tool_args_hash` records the hash of a CV lookup's arguments, the log is forensically useful without also being a second, less-governed store of applicant data.

Why `fria_ref` links artifacts instead of duplicating them. `fria_ref` holds a reference string, not the fundamental rights impact assessment itself. The event points at the FRIA document; it does not embed it. This is the same design instinct as `tool_args_hash`: the log is a spine that connects to other evidence artifacts (a FRIA, a causal chain via `causal_chain_ref`, a specific overseer via `overseer_subject_id`) rather than a single object trying to hold everything. When an auditor asks "show me the human-rights assessment behind this decision," you follow `fria_ref` to the actual document; the log does not need to carry a stale copy of it.

The other fields do what their names suggest: `system_id` and `version` are the identity fields paragraph 2(a) needs to spot a substantial modification; `session_id` and `user_subject_id` (a pseudonymous identifier, not a name) thread events belonging to one interaction together; `confidence` and `risk_flags` carry whatever signal the system itself produced about how sure or how risky a given event was; `overseer_subject_id` and `override_reason` are populated only on `review` and

`override` events, where a human's identity and reasoning are exactly what needs recording.

Tamper evidence: the hash chain, honestly

Two fields remain: `prev_hash` and `self_hash`. Together they turn a flat list of log lines into something an auditor can actually verify, not just read.

The mechanism is ordinary, not exotic. `compute_hash()` takes the event's canonical form (its fields as a dict, sorted keys, `self_hash` excluded from the computation) and runs it through SHA-256:

```
def canonical(self) -> dict[str, Any]:
    d = dataclasses.asdict(self)
    d.pop("self_hash", None)  # excluded from the hash computation
    return d

def compute_hash(self) -> str:
    body = json.dumps(self.canonical(), sort_keys=True, separators=(",", ":"))
    return hashlib.sha256(body.encode()).hexdigest()
```

`append()` looks up the `self_hash` of whatever event is currently last in the file, stamps it into the new event's `prev_hash`, computes the new event's own `self_hash`, and writes one JSON line:

```
def append(event: OversightEvent, *, path: Path = ARTIFACTS) -> OversightEvent:
    prev = _last_hash(path)
    chained = dataclasses.replace(event, prev_hash=prev)
    final = dataclasses.replace(chained, self_hash=chained.compute_hash())
    path.parent.mkdir(parents=True, exist_ok=True)
    with path.open("a") as f:
        f.write(json.dumps(dataclasses.asdict(final), sort_keys=True) + "\n")
    return final
```

Every event's hash depends on its own content and on the previous event's hash. That single design choice is what makes the log a chain rather than a pile of independent

lines. Flip one byte in an event three lines back, and that event's stored `self_hash` no longer matches what `compute_hash()` recomputes from its (now-altered) content. Worse for the tamperer: every event after it was chained to the old, correct hash, so every one of those later events also fails to verify against the (now different) chain. One edited byte breaks every subsequent link, not just the one it touched.

`verify_chain()` is the auditor's tool: it walks the file once, checks that each event's `prev_hash` matches the previous event's `self_hash`, and that each event's `self_hash` matches what its content recomputes to, collecting a list of every mismatch it finds:

```
def verify_chain(path: Path = SAMPLES) -> tuple[bool, list[str]]:
    problems: list[str] = []
    prev_hash = ""
    for i, ev in enumerate(iter_events(path)):
        if ev.prev_hash != prev_hash:
            problems.append(f"event {i}: prev_hash mismatch ...")
        recomputed = ev.compute_hash()
        if ev.self_hash != recomputed:
            problems.append(f"event {i}: self_hash mismatch ...")
        prev_hash = ev.self_hash
    return (not problems, problems)
```

It is worth naming, out loud, exactly what this protects against and what it does not, because overselling a control is worse than not building it.

What it protects against. Silent edits to an existing event. If anyone, an engineer trying to make an embarrassing decision disappear, a compromised process, a bad migration script, changes a field in a line that has already been written, `verify_chain` will find it and name the first broken link. That is real, useful protection, and it costs nothing more exotic than one SHA-256 computation per event.

> WATCH OUT
>
> **What it does not protect against, on its own.** Deleting the tail of the log. If someone truncates the file after the last event they want kept, and simply never appends the events that followed, the remaining chain still verifies perfectly: every kept event still hashes correctly to the one before it. The chain proves that nothing *within* the surviving log was altered. It cannot, by itself, prove that nothing was *removed from the end*. Defending against tail deletion needs something external to the file: shipping each new hash to a separate system the person doing the tampering does not control (a second log store, a monitoring service, even a periodically-published hash digest), or write-once storage (WORM) at the infrastructure layer that makes deletion itself impossible rather than merely detectable after the fact.

This is not a flaw specific to this schema. It is the honest limit of any hash chain that lives entirely inside one file: the chain is a property of what is there, not a guarantee about what used to be there. Production systems close that gap by anchoring the chain externally, log shipping to a separate, access-controlled store, or WORM object storage, as the complement to the in-file verification this course teaches. The schema you build here gives you the detection primitive. The shipping and anchoring is the operational discipline that makes tail deletion visible too.

The audited decorator: making logging the default path

A schema is only useful if code actually populates it, on every call, without an engineer remembering to add a log line by hand. `audit_middleware.py` wraps that discipline into a decorator:

```
def audited(
    *,
    system_id: str,
    version: str,
    model_id: str,
    fria_ref: str,
    log_path: Path | None = None,
```

```
):
    def deco(fn):
        @functools.wraps(fn)
        def inner(*args, session_id: str, user_subject_id: str,
                  confidence: float | None = None,
                  risk_flags: tuple[str, ...] = (),
                  **kwargs):
            ev = OversightEvent(
                timestamp=time.time(),
                system_id=system_id,
                version=version,
                session_id=session_id,
                user_subject_id=user_subject_id,
                event_type="tool_call",
                tool_name=fn.__name__,
                tool_args_hash=hash_args({"args": args, "kwargs": kwargs}),
                model_id=model_id,
                confidence=confidence,
                risk_flags=risk_flags,
                fria_ref=fria_ref,
                causal_chain_ref=f"chain-{session_id}",
            )
            if log_path:
                append(ev, path=log_path)
            else:
                append(ev)
            return fn(*args, **kwargs)
        return inner
    return deco
```

Applied to one of BriskDesk's tool functions, `create_return` for instance, `@audited(system_id=..., version=..., model_id=..., fria_ref=...)` emits a `tool_call` event, args hashed, before the wrapped function body runs at all. The function still has to receive `session_id` and `user_subject_id` as keyword arguments, which is a small API cost, but the payoff is that writing a new tool means the audit event comes for free. An engineer cannot ship a new BriskDesk tool that silently skips the log, short of deliberately not applying the decorator. Module 5's pre-execution checks, the ones that block a suspicious `create_return` call before it fires, land their blocked attempts in this same log as `stop` events, so the record of "what the system tried to do" and "what got stopped before it happened" live side by side.

This is the practical answer to Article 12(1)'s "technically allow." The capability is not a checklist item. It is a decorator on the function signature, and every call to that function is now a call the system technically cannot make without also logging it.

Retention: Article 19, the deployer mirror, and the sector fold-in

Recording events is half of Article 12. Keeping them is Article 19's job, and it is worth reading precisely because "at least" is the whole clause:

> THE ACT SAYS · ARTICLE 19
>
> 1. Providers of high-risk AI systems shall keep the logs referred to in Article 12(1), automatically generated by their high-risk AI systems, to the extent such logs are under their control. Without prejudice to applicable Union or national law, the logs shall be kept for a period appropriate to the intended purpose of the high-risk AI system, of at least six months, unless provided otherwise in the applicable Union or national law, in particular in Union law on the protection of personal data.
> 2. Providers that are financial institutions subject to requirements regarding their internal governance, arrangements or processes under Union financial services law shall maintain the logs automatically generated by their high-risk AI systems as part of the documentation kept under the relevant financial services law.

Three things to hold onto here, and they pull in different directions on purpose.

Six months is a floor. The text says "at least six months." It does not say six months. It says the retention period must be "appropriate to the intended purpose," and names six months as the minimum below which no intended purpose can justify going. Treating six months as a design target, rather than a floor you expect to clear comfortably, is a mistake this course flags deliberately: Article 73(6) incident investigations regularly need to look back further than six months to establish a

pattern, and a provider whose logs age out right when an investigation needs them has satisfied the letter of Article 19 while defeating its purpose.

What can lengthen it. The intended purpose of the system. A recruitment-screening system whose outputs shape years-long employment relationships has a different appropriate retention window than a system whose decisions are corrected within the hour. The provider sets this, and has to be able to justify the number chosen against what the system actually does, not against what is cheapest to store.

What can shape it. Applicable Union or national law, in particular data protection law. Article 19 explicitly subordinates itself to that law: retention cannot simply be set as long as an engineering team likes if GDPR's storage-limitation principle or a national rule says otherwise. This is not a loophole. It is the seam where the next section of this chapter, minimization inside the log itself, becomes load-bearing.

Providers keep the logs under their control. Article 19 does not reach logs a provider never had access to. That is where the deployer-side mirror in article twenty-six picks up: deployers keep the logs they control, under the same six-month floor, which is why HireSift's schema has to work for both Arbeta (the provider) and the employers who deploy it (the deployers), each holding their own slice of the trail.

Financial institutions get a fold-in rather than a parallel obligation: providers that are financial institutions maintain these logs as part of the documentation they already keep under Union financial services law, rather than standing up a second, separate retention regime. The obligation does not disappear; it gets absorbed into an existing compliance structure that already covers record-keeping at a comparable rigor.

What NOT to log: minimization inside the log

Article 12 asks you to log. GDPR (and the data-protection carve-out inside Article 19 itself) asks you to minimize what personal data you hold and for how long. These are not in tension if the schema is designed correctly, but they are in tension if it is not, and the fix has to happen at the schema level, not as an afterthought.

The vendored schema makes three minimization choices, and they are worth naming as choices rather than defaults:

Hash arguments, never store them raw. `tool_args_hash` is the mechanism already discussed above, but it is worth restating here as a minimization decision specifically: the log can prove a specific set of arguments was used (by hash comparison against a suspected value) without becoming a second, less-governed copy of whatever personal data those arguments contained.

Reference subjects by pseudonymous ID. `user_subject_id` and `overseer_subject_id` are identifiers, not names. `cust-42`, `op-alice`, not "Jane Novak." The log can still thread every event belonging to one person's interaction together, and it can still name which specific overseer reviewed or overrode a decision, without the log file itself becoming a place where a name, an email, or a CV sits in plain text.

No raw content in the audit trail. For HireSift specifically, that means no raw CV text lands in the log. The log records that a scoring event happened, references the candidate pseudonymously, and hashes the inputs that drove the decision. The CV itself lives in whatever governed record store already holds applicant data, subject to whatever retention and access rules that store already has, not duplicated into a second store with looser rules because it happened to also be a log line.

Secrets, credentials, API keys, tokens, never appear in a log line under any of these principles. That one is not a nuance; it is a bright line.

The reconciliation, in one sentence: the logging duty and the data-minimization duty do not compete for the same data, because a well-designed schema logs proof that an event happened and what governed it, not a copy of the personal data the event touched.

How logs feed Articles 72 and 73, and Module 7

This chapter opened by calling logging the evidence substrate everything else stands on. Concretely:

- **Article 72 (post-market monitoring)** needs a running baseline of normal operation to notice when something has drifted from it. The `decision` and `tool_call` events in this schema are exactly that baseline, timestamped and versioned, ready to be aggregated into the metrics Module 4's monitoring loop watches.
- **Article 73 (serious incident reporting)** needs a reconstructable timeline: what the system did, in what order, who was overseeing it, and whether anyone intervened. That is a `verify_chain` walk followed by a filter over `event_type`, not a fresh investigation built from scratch.
- **Module 7's oversight measurement** asks whether humans reviewing a system's outputs actually catch its failures, not just whether a review step exists on paper. That question is answered by comparing `decision` events against the `review` and `override` events that follow them, which is only possible because both event types live in the same schema, chained together, with `overseer_subject_id` recording who did the reviewing.

None of these downstream duties work if the log schema was designed as an afterthought, one field bolted on whenever a specific requirement was pointed at. Designing the schema once, against the taxonomy and the fields this chapter has walked through, is what makes the other three obligations a query against existing data instead of a new project.

Worksheet: design VibraSense's logging schema

Norrfelt Industrial AB's VibraSense is not high-risk under the current classification (Module 2 covered why: it schedules maintenance, it never actuates anything). Article 12 does not bind it. But VibraSense still produces decisions that matter, a false alarm costs technician time, a missed alert costs a failed pump, and Norrfelt's Article 6(4) documentation memo is stronger for including a logging section that shows the team thought about evidence before a regulator or a customer ever asked for it.

Using what you know about VibraSense (a gradient-boosted classifier scored nightly in batch over vibration and temperature sensor time series, alerting a service-planning

team rather than controlling any machine), design its logging schema. Cover:

1. Which events VibraSense should log, using the six-type taxonomy from this chapter (`tool_call`, `decision`, `review`, `override`, `stop`, `system change`) as your starting point. Not every type will apply, or will apply the same way it does for HireSift or BriskDesk.
2. What retention period you would set, and how you would justify it against the six-month Article 19 floor even though VibraSense sits outside Article 19's scope.
3. What the memo should say, given VibraSense is not high-risk today, about why Norrfelt logs anyway.

Write your answer before reading the appendix. The value of the exercise is in the choices you make before you see someone else's.

Appendix: sample answer

Events logged. VibraSense's nightly scoring run is a `decision` event: one per pump, per run, recording the failure-risk score, the model version that produced it, and a reference to the sensor-data window it was computed over (mirroring `causal_chain_ref`'s role of linking a decision back to its inputs, the same way HireSift links a ranking decision to a specific applicant pool snapshot). When the score crosses the alert threshold and a service visit gets scheduled, that scheduling action is a `tool_call` event against Norrfelt's service-planning system. When a technician inspects a flagged pump and finds nothing wrong, or finds something the model missed, that inspection outcome is a `review` event, and if it leads someone to override the model's recommendation, an `override` event with the technician's reasoning recorded. `system change` events fire whenever the model is retrained on a new batch of fleet data, which matters directly for catching the sensor-aging drift this course's Module 4 chapter already flagged as VibraSense's main robustness risk. `stop` rarely applies today, since VibraSense only recommends and never actuates, but the field stays in the schema precisely because the classification could flip if a future version is wired to derate or stop a pump, and a schema built without a `stop` type would need retrofitting exactly when the stakes rose.

Retention. Norrfelt sets retention at at least the same six-month floor Article 19 would require if VibraSense were high-risk, and in practice keeps it closer to the multi-year horizon a bearing's service life spans: a false-negative failure (the model missed a real bearing failure) may only become obvious months after the fact, when the pump actually fails, and by then a six-month log would already have aged out the very decision events that explain what the model saw and predicted at the time. The memo states this explicitly: the floor Article 19 sets for high-risk systems is treated as the minimum bar even for a system outside that article's scope, because the operational reason for keeping logs (explaining a failure after the fact) does not care what legal tier the system sits in.

Why log anyway. The memo's honest answer is that "not high-risk" is a conclusion about legal obligations, not a conclusion about whether evidence is useful. A false alarm has a cost, a missed failure has a cost, and a logging schema is what lets Norrfelt tell, after either one happens, whether the model made a defensible call on the data it had. It is also insurance against the classification changing: if a future VibraSense version is wired to derate or stop a pump, it crosses into a use-case area Article 12 would very plausibly bind, and Norrfelt would rather already have the taxonomy, the retention practice, and the habit of hashing sensitive inputs in place than build all three under deadline pressure the week the classification flips.

CHAPTER 7

Human Oversight: Article 14 as a Measurable System Property

Every compliance deck has a slide with a human icon standing next to an AI icon, an arrow between them, and a caption that says “human in the loop.” Almost none of those decks say how anyone knows the human catches anything. A reviewer who rubber-stamps every output is, on paper, exactly as present as a reviewer who reads carefully and flags real problems. The org chart cannot tell them apart. Article 14 of the AI Act can, because it does not ask for a human in a box. It asks for a human who can **effectively** oversee the system, and effectiveness is the kind of word that has a number attached to it once you decide to measure it.

This module builds that number. It reads Article 14 the way Module 4 read Article 15: as a specification, not a mood. It names the cognitive failure mode the Act itself names. It borrows a measurement design from Robert’s own `vigil` framework and walks the exact metrics that design produces. And it works through a real, bundled review session, three reviewers, thirty items, ten planted failures, so that “measure oversight” stops being an abstraction and becomes a confusion matrix you can compute yourself.

Reading Article 14 as a spec

Here is the text, unmodified, from Chapter III Section 2 of the enacted Act:

THE ACT SAYS · ARTICLE 14

1. High-risk AI systems shall be designed and developed in such a way, including with appropriate human-machine interface tools, that they can be effectively overseen by natural persons during the period in which they are in use.
2. Human oversight shall aim to prevent or minimise the risks to health, safety or fundamental rights that may emerge when a high-risk AI system is used in accordance with its intended purpose or under conditions of reasonably foreseeable misuse, in particular where such risks persist despite the application of other requirements set out in this Section.
3. The oversight measures shall be commensurate with the risks, level of autonomy and context of use of the high-risk AI system, and shall be ensured through either one or both of the following types of measures: (a) measures identified and built, when technically feasible, into the high-risk AI system by the provider before it is placed on the market or put into service; (b) measures identified by the provider before placing the high-risk AI system on the market or putting it into service and that are appropriate to be implemented by the deployer.

> 4. For the purpose of implementing paragraphs 1, 2 and 3, the high-risk AI system shall be provided to the deployer in such a way that natural persons to whom human oversight is assigned are enabled, as appropriate and proportionate: (a) to properly understand the relevant capacities and limitations of the high-risk AI system and be able to duly monitor its operation, including in view of detecting and addressing anomalies, dysfunctions and unexpected performance; (b) to remain aware of the possible tendency of automatically relying or over-relying on the output produced by a high-risk AI system (automation bias), in particular for high-risk AI systems used to provide information or recommendations for decisions to be taken by natural persons; (c) to correctly interpret the high-risk AI system's output, taking into account, for example, the interpretation tools and methods available; (d) to decide, in any particular situation, not to use the high-risk AI system or to otherwise disregard, override or reverse the output of the high-risk AI system; (e) to intervene in the operation of the high-risk AI system or interrupt the system through a 'stop' button or a similar procedure that allows the system to come to a halt in a safe state.
> 5. For high-risk AI systems referred to in point 1(a) of Annex III, the measures referred to in paragraph 3 of this Article shall be such as to ensure that, in addition, no action or decision is taken by the deployer on the basis of the identification resulting from the system unless that identification has been separately verified and confirmed by at least two natural persons with the necessary competence, training and authority. The requirement for a separate verification by at least two natural persons shall not apply to high-risk AI systems used for the purposes of law enforcement, migration, border control or asylum, where Union or national law considers the application of this requirement to be disproportionate.

Five paragraphs, each doing a different job. Paragraph 1 sets the target, and it is a strong word: **effectively** overseen, not merely accompanied by a human, with the interface tools that make effective oversight possible named as part of the design obligation, not an afterthought. Paragraph 2 says what oversight is for: preventing or

minimising risk under both the intended use and reasonably foreseeable misuse, which rules out the excuse “nobody was supposed to use it that way.” Paragraph 3 splits the work between two parties: measures the provider builds into the system itself (3(a)) and measures the provider identifies but hands to the deployer to implement (3(b)). The split recurs through the whole module: the provider designs the levers, the deployer staffs them. Arbeta GmbH, HireSift’s provider, builds the interface that shows a recruiter what the ranking model saw and why, and tells the deployer employer what measures it needs to run; the employer, as deployer, actually assigns a competent person to the shortlist notes and gives that person the time to look properly.

Paragraph 4 is the enablement list, and it reads like a checklist a software team could actually implement, because it is one. Five capabilities, each mapping cleanly onto an engineering deliverable: understanding limits maps to a model card and training material; monitoring for anomalies maps to the logging and monitoring views this course built in Module 6; correctly interpreting output maps to interpretation tools and confidence displays; deciding to disregard, override, or reverse maps to an override control, one that is logged, not silent; and intervening or stopping maps to a stop button reaching a safe halt state. Paragraph 4(b) is the one this chapter spends the most time on, because it is the paragraph that turns “human in the loop” from decoration into an empirical claim, and the next section takes it on its own.

Paragraph 5 adds a hard rule for the single highest-stakes Annex III category: remote biometric identification. This chapter returns to it after the measurement design, because the two-person rule is the Act’s own answer to the question this whole module is asking: how much do you trust one person’s judgment on a consequential call.

Oversight architectures, with their honest costs

Article 14 does not mandate one shape of oversight. It asks for measures commensurate with risk, autonomy, and context, which in practice means different systems in the cast earn different architectures.

ARCHITECTURE	WHAT IT MEANS	CAST EXAMPLE	COST	FAILURE MODE
Human-in-the-loop	A person approves before the output takes effect	HireSift's recruiter approves a shortlist note before it reaches a hiring manager	Highest: someone reviews every item, which does not scale past a certain volume	The review becomes a formality if the volume outpaces the time budget
Human-on-the-loop	A person monitors a running system and can intervene, but does not approve every action	VibraSense's service planner watches the failure-risk score and can override a scheduled visit	Lower: monitoring is cheaper than per-item approval	The planner can simply ignore the score and never intervene, and nothing in the architecture forces attention
Human-in-command	A person can stop the whole system, a broader authority than intervening in one decision	The Module 6 stop event, which halts a system pending investigation	Rare to invoke, but must be reachable at all times	If the stop path is undocumented or slow to reach, it exists on paper only
Machine-assisted oversight	A second automated system pre-screens for the human, narrowing what the human sees	Module 5's judge layer flags likely-problem outputs before a human reads them	Reduces reviewer load	The assistant becomes the de facto decision-maker if the human defers to it, which is automation bias one layer up

The last row deserves a second look, because it is the one teams reach for first and reason about the least. An LLM-as-judge layer that pre-screens outputs for a human reviewer is a genuine efficiency gain: instead of reading every item cold, the reviewer starts from a triaged queue. But the judge is itself a system a human can over-rely on, and if its own reliability has not been measured (Module 4 covered how to measure judge agreement, position bias, and self-preference), the oversight layer built on top of it inherits an unmeasured foundation. Machine-assisted oversight is an assist, and the word "assist" only holds if the human retains and exercises the authority to disagree with it.

Automation bias: the phenomenon the Act names by name

Article 14(4)(b) does something unusual for a piece of legislation: it names a specific cognitive failure mode inside the operative text, in parentheses, as if daring a reader to look it up. "To remain aware of the possible tendency of automatically relying or over-relying on the output produced by a high-risk AI system (automation bias)." That parenthetical is not decoration. A law that names a bias by name is not describing a hypothetical; it is telling providers and deployers that this specific failure has already been observed often enough, in systems that hand information or recommendations to a human decision-maker, that the drafters felt it needed a label in the statute itself. The clause is explicit that the risk is sharpest "for high-risk AI systems used to provide information or recommendations for decisions to be taken by natural persons," which describes HireSift's shortlist notes exactly: a recruiter reading an AI-written summary, deciding whether to trust it, under time pressure, batch after batch.

This course does not cite an academic literature on automation bias, because doing so would dress up a well-known phenomenon in borrowed authority it does not need. The citation that matters for a compliance engineer is the one already sitting in the article: the Act's own text names the tendency and requires the deployer to keep the overseer aware of it. What the Act does not tell you is how to know whether awareness translated into actual catches. That is a measurement problem, and the rest of this chapter builds the measurement.

The closed-loop measurement design, credited to vigil

The measurement design in this module is not something the course invented. It is the closed-loop pattern from Robert's own `vigil` framework (github.com/robertbarcik/vigil), a red-teaming and human-oversight measurement tool, applied here to Article 14 evidence instead of vigil's original red-team use case. The idea is simple to state and does real work once it is running:

1. Take a stream of items a human reviewer will look at, HireSift's AI-written shortlist summaries in this course's case.
2. Plant some known-bad items into the stream. Vigil calls these probes: items with a real, documented issue (`has_issue = true`), tagged with an issue type and description, mixed in among ordinary, clean items.
3. Give the mixed stream to a human reviewer with no indication of which items are planted.
4. Score the reviewer against the ground truth you planted, not against a proxy like throughput or time-on-task.

Scoring against planted ground truth is the entire point: a review queue with no known-bad items gives no way to tell a careful reviewer from a rubber stamp, because both produce the same "no issues found" output on a stream of genuinely clean items. Probes turn an unfalsifiable claim ("our reviewers catch problems") into a falsifiable one you can test.

Vigil's `OversightSession` schema produces three core metrics per reviewer, and this course computes the identical metrics with pandas on a bundled dataset rather than the live vigil pipeline:

- **Detection rate**: of the items with a real, planted issue, what fraction did the reviewer flag. This is recall against ground truth, not recall against the reviewer's own sense of how many problems exist.
- **Precision**: of everything the reviewer flagged, what fraction was a real issue. A reviewer who flags everything gets perfect detection and terrible precision, which is exactly the failure mode a naive "did they catch it" metric would miss.

- **Response time**: how long the reviewer took per item, in seconds.

Those three combine into a single vigilance score:

```
vigilance = 0.5 * detection + 0.3 * precision + 0.2 * speed
speed = max(0, 1 - avg_time_seconds / 120)
```

The weighting is a judgment call, worth stating plainly rather than presenting as derived from the Act, because the Act specifies none of this. Detection gets the largest share, one half, because a reviewer who never catches anything provides no oversight regardless of how fast or careful their flags look. Precision gets three tenths, enough to penalise a reviewer who "catches everything" by flagging indiscriminately, but not enough to swamp detection. Speed gets only one fifth, deliberately: a fast reviewer with mediocre detection and precision should not be able to outscore a slow, accurate one on throughput alone. The 120-second cap and the floor at zero are also choices, not physics: past two minutes per item, additional slowness buys nothing more, and speed can never drag a score below what detection and precision already earned. None of these weights were audited by anyone but the person who wrote them, and the same caveat belongs in your own evidence pack: state your weights, state why, and expect an auditor to ask.

The live path, for anyone who wants to run the actual vigil pipeline rather than the bundled data, is `pip install vigil` (it pulls in vigil's FastAPI stack as a dependency), `vigil demo load` to fetch a demo run, `create_closed_loop_session` on that run's transcripts, and `vigil serve` for a web review UI. Worth saying honestly: the packaged demo run ships only two clean transcripts, so a closed-loop session built directly from it either forces a very low issue threshold or turns out degenerate, with almost nothing to review. That is a real limitation of the demo data, not a setup bug, and the bundled `oversight_session.json` this module uses exists precisely so the exercise has a properly sized, thirty-item session instead.

Three reviewers, one session: the worked example

The course ships `notebooks/data/m07/oversight_session.json`, a fictional dataset authored for this course in vigil's `OversightSession` schema. It is not real hiring data and no real candidates are involved; every name in it is invented and every issue was planted on purpose. The session simulates thirty HireSift shortlist-summary review items, ten of which carry a planted issue: fabricated claims (a certification or degree the CV never mentions), biased language (evaluative wording coded to age or origin), wrong data (an employment record copied from a different candidate in the batch), and omitted information (a gap or missing skill the summary quietly drops). Five of the planted issues sit in the first half of the queue, five in the second, which is what makes the fatigue analysis in the next section possible. Three fictional reviewers each made thirty decisions against the same items, with response times recorded per item. The session's `scores` field ships empty on purpose: computing detection, precision, and vigilance is the notebook's job, not something handed to the student pre-baked.

Working the confusion matrix by hand for each reviewer gives three sharply different profiles.

REVIEWER	DETECTION	PRECISION	AVG. RESPONSE TIME	WHAT AN AUDITOR CONCLUDES
rev-thorough	0.90 (9 of 10 planted issues caught)	1.00 (zero false flags)	~80 seconds	Slow, but every flag is real and the miss rate is low. This is the reviewer you can show an auditor as evidence of effective oversight.

REVIEWER	DETECTION	PRECISION	AVG. RESPONSE TIME	WHAT AN AUDITOR CONCLUDES
rev-fast	0.50 (5 of 10 caught)	0.50 (5 false flags alongside 5 real ones)	~12 seconds	Fast, but half the flags are noise and half the real issues slip through. A throughput dashboard would call this reviewer excellent; the ground truth says the opposite.
rev-fatigue	0.60 overall, but split 5-of-5 early against 1-of-5 late	roughly 0.86 (six of seven flags were real)	speeds up from roughly forty seconds early to about twenty-two seconds late	Perfect in the first half of the session, close to blind in the second. The aggregate number hides the story entirely.

Feeding the formula, rev-thorough's vigilance lands around 0.82 (0.5 x 0.90 + 0.3 x 1.00 + 0.2 x roughly 0.33 for an 80-second average), and rev-fast's lands around 0.58 (0.5 x 0.50 + 0.3 x 0.50 + 0.2 x roughly 0.90 for a 12-second average). rev-fast's speed component is the highest of the three, nearly full marks, since twelve seconds per item is fast against the 120-second cap. That is the gotcha this table surfaces: on any dashboard that measures throughput, rev-fast looks like the best-performing reviewer in the queue, and by the metric that actually matters, whether real problems get caught, rev-fast is the worst overseer of the three.

rev-fatigue's whole-session vigilance computes to roughly 0.70, sitting between the other two and, taken alone, looking unremarkable. That number is the clearest argument in this chapter for why an aggregate score is not enough on its own: 0.70 says nothing about the shape of the failure underneath it. A reviewer who is consistently mediocre and a reviewer who is perfect for fifteen items and then nearly blind for

fifteen more can land on the same aggregate number. Only the early-versus-late split exposes which one you actually have.

Fatigue and probes

Vigilance decays within a session, and rev-fatigue's split, five of five planted issues caught in the first half of the queue against one of five in the second, is the cheapest fatigue detector this module has: split the session at its midpoint and compare detection before and after. No new instrumentation is required beyond the timestamp already on every decision. A reviewer whose detection collapses in the back half of a batch is not careless as a personal failing; it is a predictable property of sustained attention, and the fix belongs to the process, not a lecture to the reviewer. Rotation (nobody works an entire queue alone), batch limits (cap how many items one person reviews before a break), and probe cadence (keep planting known-bad items throughout the session, not only at the start) are the operational knobs this pattern points to.

Probes carry a second, quieter function beyond catching fatigue: they make measurement possible at all in production. Ground truth on a live review queue does not exist by default, since an item you already know has an issue would not need a human reviewer to find it. Planted probes, refreshed periodically and rotated so reviewers cannot memorise which items are probes, are how a real deployment keeps its own oversight measurement honest over time, not only at the one moment a course dataset was assembled.

The two-person rule and its limits

Article 14(5) sets a hard floor under all this measurement, for exactly one category: systems under Annex III point 1(a), remote biometric identification. For those systems, the deployer may not act on an identification the system produces unless at least two competent, trained, and authorised natural persons have separately verified it. Two people, checking independently, before consequential action, is the Act's own answer to the question this whole chapter has been building toward with metrics: how much should you trust one person's judgment on the highest-stakes calls. The honest carve-

out belongs in the same breath: the second paragraph of 14(5) removes the two-person requirement for law enforcement, migration, border control, and asylum uses, where Union or national law judges the requirement disproportionate. That carve-out exists in the enacted text; it is not a gap this course invented, and it is also not a detail to bury.

> **WATCH OUT**
>
> It is worth being precise about what 14(5) does and does not bind. HireSift sits in Annex III point 4(a), recruitment and selection, not point 1(a), so the two-person rule is not a legal requirement on HireSift's shortlist review. What HireSift can borrow is the pattern, not the mandate: a recruiter's rejection driven by an unusually low or anomalous ranking score is exactly the kind of consequential, hard-to-reverse decision that benefits from a second competent person's independent look before it becomes final, the same logic 14(5) applies by law to biometric identification. Calling this an analogy rather than an obligation matters, since overstating a legal requirement is its own kind of dishonesty, and an auditor, or a plaintiff's lawyer, will draw the distinction between "the law requires this" and "good practice modelled on what the law requires elsewhere" immediately if you do not draw it first.

Where oversight events land: the Module 6 log

Every measurement in this chapter needs somewhere to live once a real HireSift is running, and that somewhere is the hash-chained oversight log this course built in Module 6. Two fields in that log's `OversightEvent` schema exist specifically for Article 14: `overseer_subject_id`, naming the natural person who acted, and `override_reason`, a free-text field empty for ordinary tool calls and decisions but required to carry real content the moment `event_type` is `"override"` or `"stop"`. An override event with a blank `override_reason` is a broken record, not a minor omission: paragraph 4(d), the right to disregard, override, or reverse the system's output, only becomes evidence once the reason for exercising it is written down somewhere an auditor can read later. The same log's `"review"` event type is where a quarterly Article 14 audit gets recorded: how many sessions were sampled, whether the

hash chain verified intact, whether every override in the sample carried a reason. Oversight measurement is not a side notebook the compliance team runs once; it is a query over the same append-only log Module 6 built for every other Article 12 purpose, filtered down to the event types this article cares about.

Building oversight_evidence.md

The hands-on notebook for this module, `m07_oversight_evidence.ipynb`, takes the bundled session, computes the per-reviewer confusion matrix, detection, precision, and vigilance, splits the session at item fifteen for the fatigue check, and renders all of it into `outputs/oversight_evidence.md`, a document shaped directly around Article 14(4)'s five-item list. The rendered evidence file states, for each of 14(4)(a) through (e), what measure is in place (a monitoring view, an override control, a stop button), what the measured effectiveness actually is where a number exists (rev-thorough's 0.90 detection, rev-fast's 0.50 precision, rev-fatigue's early-versus-late split), what gaps remain (rev-fast is not an oversight profile you would want staffing a live queue; rev-fatigue needs a rotation policy), and what the review cadence is going forward. That is the artifact you hand an auditor instead of a slide with a human icon on it.

Worksheet: design BriskDesk's escalation oversight

BriskDesk, Solvana's customer-support chatbot, is not a high-risk system and Article 14 does not bind it. It still escalates conversations to a human agent when it cannot resolve a request or when a customer asks for one, and that escalation path is, in miniature, exactly the oversight problem this chapter has been measuring: a human is supposed to catch what the automated system got wrong or could not handle, and nobody has checked whether that human actually does.

Design the oversight scheme for BriskDesk's escalation path. Cover three things:

1. What does the human agent see at the moment of escalation? Not just the customer's last message: what context, what confidence signal, what prior turns, would let a human actually judge whether the bot's handling up to that point was reasonable.

2. What would you measure, and how would you plant ground truth into an escalation queue the way this module planted issues into HireSift's review queue? What would a "planted probe" even look like for a support handoff.
3. What is the stop condition? At what point does a pattern of escalations (a spike, a specific failure type recurring) trigger something bigger than one agent's individual response, the way Module 6's kill-switch event halts an entire flow.

Write your own answer before reading the appendix.

Appendix: sample answer

What the agent sees. The escalation handoff carries the full conversation transcript, not a summary, plus BriskDesk's own confidence signal on the last few turns (retrieving from the product catalog with a strong match, or falling back on a generic response), plus a one-line reason code for why the handoff triggered (customer requested a human, retrieval confidence dropped below threshold, or the message matched a trigger like a refund dispute). An agent reading only the last message has no way to judge whether BriskDesk mishandled the previous turns; the transcript plus the confidence trace turns the handoff into something reviewable rather than a cold start.

What gets measured, and how to plant probes. The equivalent of a planted issue here is a periodically injected test conversation, scripted by Solvana's own QA team, run through the same escalation path as real traffic, with a known correct outcome (this conversation should escalate; this one should not; this refund request should be flagged, not auto-approved). Mixing a small number of these into the real queue, without telling agents which conversations are scripted, gives Solvana the same closed-loop measurement built for HireSift: detection, precision, and response time, scored against ground truth Solvana controls because it wrote the test cases itself.

The stop condition. A single agent's occasional bad call is a training issue, not a system issue. The stop condition sits one level up: if the escalation rate for a specific failure type spikes across many customers in a short window, or retrieval confidence on a whole class of product questions collapses at once, that pattern indicates something changed in BriskDesk itself, a poisoned catalog entry, a broken retrieval index, an

upstream model update behaving differently, not a run of unlucky conversations. That should trip a stop event the same shape as Module 6's kill-switch: halt the affected flow, route everything to human handling until someone investigates, and log the stop with a reason, exactly the way `override_reason` and `overseer_subject_id` already require for HireSift's oversight log.

CHAPTER 8

Technical Documentation Without Tears: Article 11 and Annex IV

Module 7 asked whether a human overseer actually catches what an AI system gets wrong. This module asks a duller question that turns out to matter just as much: can you prove, to someone who was not in the room, that the first seven modules happened at all. Article 11 and its companion Annex IV are the Act's answer to that question, and they have a bad reputation among developers before anyone has read a word of them. Annex IV runs to nine points and reads like a checklist assembled by a committee that never had to fill one out.

The thesis of this chapter is that the reputation is earned only if you treat documentation as something written the month before a conformity assessment. Read the nine points carefully and a pattern appears: point 1 is the classification memo and model card from Module 2. Point 2's data requirements are the datasheet and bias work from Module 3. Point 2's validation section and point 3's accuracy figures are the declared metrics from Module 4. Point 2's cybersecurity section is the security-testing evidence from Module 5. Point 6's lifecycle changes are the system-change log Module 6 designs. Point 2's human-oversight assessment and point 3's oversight measures are the evidence Module 7 produces. By the time you reach this module, most of Annex IV already exists somewhere in a folder. What is missing is the assembly. That is the actual skill this chapter teaches: not writing technical documentation from a blank page, but building the machine that assembles it from artifacts you already have, and being honest, in writing, about the handful of sections nothing has produced yet.

Reading Article 11(1) as a spec

Here is the operative sentence, unmodified, from Chapter III Section 2 of the enacted Act:

> THE ACT SAYS · ARTICLE 11(1)
>
> The technical documentation of a high-risk AI system shall be drawn up before that system is placed on the market or put into service and shall be kept up-to date.

Two duties in one sentence, and both are absolute. “Drawn up before” placement on the market rules out documentation as a post-hoc artifact assembled to satisfy an auditor’s request after the system is already live. “Kept up-to date” rules out documentation as a one-time deliverable at all; it is a living record that has to track the system for as long as the system is on the market.

The article continues, and the rest matters just as much. The documentation must be drawn up “in such a way as to demonstrate” compliance with Chapter III Section 2 and to give “national competent authorities and notified bodies… the necessary information in a clear and comprehensive form to assess the compliance of the AI system with those requirements.” It must contain, at a minimum, the elements set out in Annex IV. Then the SME provision: “SMEs, including start-ups, may provide the elements of the technical documentation specified in Annex IV in a simplified manner. To that end, the Commission shall establish a simplified technical documentation form targeted at the needs of small and microenterprises.” Where an SME opts to use it, it “shall use the form referred to in this paragraph,” and “notified bodies shall accept the form for the purposes of the conformity assessment.” There is also an Article 11(2) provision worth knowing even though it does not touch our cast: where a high-risk system is a component of a product covered by the Union harmonisation legislation in Annex I Section A, a single combined set of technical documentation covers both the AI Act’s requirements and that product legislation’s, rather than two parallel dossiers.

The obligations bind Annex III high-risk systems from 2 December 2027, the date the 2026 Digital Omnibus fixed. Of the cast, only HireSift, Arbeta GmbH’s CV screener, carries this duty as a matter of law. BriskDesk is limited-risk and Annex IV does not apply to it at all, though a slim model card is still how Solvana answers a client’s due-diligence questionnaire, a habit worth keeping regardless of legal tier. VibraSense sits outside Annex III on today’s facts, and Norrfelt’s Article 6(4) memo is not an Annex IV

document, though the same discipline of writing things down before someone asks protects Norrfelt if that classification ever flips.

The nine points of Annex IV, mapped to where they come from

Annex IV is titled, precisely, “Technical documentation referred to in Article 11(1),” and it opens with “The technical documentation referred to in Article 11(1) shall contain at least the following information, as applicable to the relevant AI system.” Nine numbered points follow. Below, each point gets three things: what it actually asks for, which module’s artifact in this course carries it, and one worked line showing what it looks like for HireSift.

Point 1: General description. Asks for the intended purpose, the provider’s name, the version and its relation to previous versions; how the system interacts with hardware or software including other AI systems; relevant software or firmware versions and update requirements; the forms in which it is placed on the market (embedded software, downloads, APIs); the hardware it runs on; product photographs where it is a component of a physical product; a basic description of the deployer’s user interface; and instructions for use for the deployer. This point comes from the Module 2 classification memo (intended purpose, provider identity) plus the model card (system identity and version) plus the integration facts you already know about how the system is embedded. For HireSift: intended purpose is CV parsing, scoring, ranking, and shortlist-note generation inside an applicant-tracking system; provider is Arbeta GmbH; the system is placed on the market as a licensed module embedded in client ATS deployments, not sold standalone; there is no physical product, so point 1(f) does not apply; the deployer UI is the recruiter-facing ranking dashboard, and the instructions for use tell a recruiter what the shortlist score means and does not mean.

Point 2: Detailed description of development. The longest point, covering methods and steps of development including recourse to pre-trained third-party tools (2(a)); design specifications, general logic, key design choices, what the system optimises for, and trade-off decisions (2(b)); system architecture and the computational resources used to develop, train, test, and validate (2(c)); data requirements as datasheets covering provenance, scope, characteristics, labelling, and cleaning (2(d));

the human-oversight assessment under Article 14 (2(e)); pre-determined changes (2(f)); validation and testing procedures, metrics, and dated and signed test logs (2(g)); and cybersecurity measures (2(h)). This point draws from more modules than any other: 2(a) from the Module 2 memo (which pre-trained model the LLM summarizer calls, and how Arbeta's prompt template constrains it); 2(d) from Module 3's datasheet on the ten years of historical hiring outcomes; 2(e) from Module 7's oversight assessment; 2(g) from Module 4's declared metrics and dated test logs; 2(h) from Module 5's security-testing evidence. For HireSift: 2(a) names the hosted model behind the shortlist summarizer and states it is used unmodified through an API, not fine-tuned; 2(d) points straight at the datasheet describing the ten years of client hiring outcomes, how they were collected, and the historical-bias caveats Module 3 flagged; 2(g) attaches the dated, signed declared-metrics report from Module 4 alongside the security-test log from Module 5.

Point 3: Monitoring, functioning, and control. Capabilities and limitations, including "the degrees of accuracy for specific persons or groups of persons" and the overall expected accuracy; foreseeable unintended outcomes and sources of risk; the human-oversight measures needed under Article 14; and input-data specifications. This comes from Module 4's declared metrics and Module 7's oversight measures. For HireSift, point 3 states precision at the shortlist cutoff and recall of qualified candidates as the overall figures, then reports the same two numbers broken out by the protected-characteristic subgroups Module 3 examined, naming the true-positive-rate gap directly rather than averaging it away into a single headline number.

Point 4: Appropriateness of the performance metrics. A description of why the chosen metrics fit this specific system. This is Module 4's justification, not its numbers. For HireSift, point 4 explains why a bare accuracy percentage would mislead for a ranking task and why precision at the cutoff, recall of qualified candidates, and the subgroup gap are the three figures that actually describe what can go wrong when HireSift filters a pool of real applicants.

Point 5: Risk management system under Article 9. A detailed description of the continuous, iterative risk-identification and mitigation process required by Article 9. No module in this course builds a standalone Article 9 risk management system, so this is a

section the assembler leaves as an honest TODO rather than papering over. For HireSift, the skeleton reads: “Article 9 risk management system: not documented by this evidence pack; needs a standalone record of risk identification, evaluation, and mitigation across the ranking model’s lifecycle.”

Point 6: Lifecycle changes. A description of relevant changes the provider made to the system over its lifetime. This is Module 6’s system-change log, nothing more exotic. For HireSift, point 6 pulls directly from the change events Module 6’s logging schema already records: every ranking-model retrain, every edit to the summarizer’s prompt template, each with a timestamp and a stated reason.

Point 7: Harmonised standards, or a description of the solutions adopted. Either a list of applied harmonised standards published in the Official Journal, or, where none exist, “a detailed description of the solutions adopted to meet the requirements set out in Chapter III, Section 2.” As of this course, zero AI Act harmonised standards are cited in the Official Journal, so point 7 is never a citation list today; it is the description of solutions actually adopted. For HireSift, that description is the evidence pack itself: the bias-examination method from Module 3, the declared-metrics and eval-pipeline approach from Module 4, the security-testing method from Module 5, stated plainly as the solutions Arbeta chose in the absence of an official standard to point to.

Point 8: Copy of the EU declaration of conformity. The signed declaration under Article 47. Nothing produces this before a conformity assessment actually happens, so it stays a TODO in the skeleton until Arbeta completes one. Naming the gap honestly is better than inserting a placeholder that reads as filled.

Point 9: Post-market monitoring plan. A detailed description of the system for evaluating performance in the post-market phase under Article 72, including the plan referred to in Article 72(3). This is deliberately Module 9’s job, not this module’s. The skeleton points at it rather than guessing at a plan this module has no basis to write.

Three points fill from artifacts you already have (1, 2 mostly, 3, 4, 6, 7). Three points stay honest TODOs until later work exists (5, 8, 9). That split is not a flaw in the course;

it is the accurate state of an Annex III system built through Module 7 and not yet through conformity assessment.

Three phrases worth reading twice

Three phrases in Annex IV are easy to skim past and expensive to skim past. Read them slowly.

Point 2(b) asks for "the key design choices including the rationale and assumptions made" and, later in the same point, "the decisions about any possible trade-off made regarding the technical solutions adopted to comply with the requirements set out in Chapter III, Section 2." Read together, this is Annex IV asking for your architecture decision records. Not the code, the reasoning behind the code: why HireSift's team chose a gradient-boosted ranker over a linear scorer, why the summarizer runs through a hosted API instead of a fine-tuned in-house model, what got traded away to hit a latency target. If that reasoning lives only in a Slack thread or a departed engineer's memory, point 2(b) is not satisfied no matter how good the code is.

> **WATCH OUT**
>
> Point 2(g) asks for "test logs and all test reports dated and signed by the responsible persons." This is not a stylistic flourish. "Dated and signed" turns CI output into an attributable record: not just that a test suite ran, but when, and who stood behind the result. A green CI badge with no date and no named owner is not what this clause is asking for; a declared-metrics report with a run date and a named reviewer is.

Point 3 asks for capabilities and limitations "including the degrees of accuracy for specific persons or groups of persons on which the system is intended to be used." Subgroup accuracy is not an optional nicety layered on top of an overall number; Annex IV names it directly, in the same breath as the overall figure. The subgroup work Module 3 does on HireSift's protected characteristics is not garnish on the bias chapter. It is Annex IV point 3, in the text.

The SME simplified form, honestly

Article 11(1) promises something specific: a Commission-established simplified technical documentation form for SMEs and start-ups, one that notified bodies "shall accept… for the purposes of the conformity assessment." That promise is unconditional in the text.

> AS OF MID-2026
>
> As of early July 2026, the form does not exist. No Commission-published simplified technical documentation form is available for an SME to actually use.

This is worth saying plainly rather than working around it. If you run a small HR-tech vendor and want to use the simplified route Article 11(1) describes, there is currently nothing official to pick up and fill in. Unofficial stopgaps exist, such as ECNL's guide, and they can be useful for orientation, but they are not the Commission's form, carry no notified-body acceptance guarantee, and should be labeled clearly as unofficial wherever you reference them.

The practical move, and the one this module's exercise follows, is to build the full Annex IV structure now, with honest TODOs where nothing exists yet, and simplify later once the official form lands. A full structure with visible gaps is easier to trim down than a simplified sketch is to expand under deadline pressure, and it means the day the Commission publishes the form, you are mapping an existing, complete document onto it rather than starting over.

Model card as substrate, Annex IV as superset

A Hugging Face-style model card is the closest thing the ML community already has to Annex IV in miniature, and it is worth being precise about what it covers well and what it leaves out. A good model card typically states intended use, in-scope and out-of-scope behaviour, headline metrics, and a summary of the training data. That maps cleanly onto part of Annex IV point 1 and part of point 2(d). It does not, on its own, cover the trade-off rationale point 2(b) asks for, the computational-resources figure in point 2(c),

dated and signed test logs under 2(g), the Article 14 oversight assessment in 2(e), the post-market monitoring plan in point 9, or a declaration of conformity under point 8. A model card is necessary. It is not sufficient.

The vendored `ModelCard` dataclass in `code/governance/governance/model_card.py` is this course's version of that substrate:

```
@dataclass
class ModelCard:
    system_id: str
    version: str
    intended_use: str
    domain: str
    in_scope: list[str] = field(default_factory=list)
    out_of_scope: list[str] = field(default_factory=list)
    model_id: str = "openai/gpt-4o-mini"
    cost_ceiling_usd_per_day: float = 0.0
    tools_allowed: list[str] = field(default_factory=list)
    risk_classification: str = "limited"
    fria_ref: str = ""
    kill_switch_url: str = ""
    oversight_schema_version: str = "1.0"
    retention_days: int = 180
    contacts: dict[str, str] = field(default_factory=dict)
```

`write_card` serialises one of these to JSON, and the point of running it at build time rather than writing a card by hand once is the same point Module 7 made about oversight configuration: a card generated by code cannot silently drift from what actually deployed, because the deploy pipeline reads the same object.

Mapping the fields onto Annex IV: `system_id`, `version`, `intended_use`, and `domain` feed point 1's general description directly. `in_scope` and `out_of_scope` sharpen point 1's instructions for use and point 3's capabilities-and-limitations language. `model_id` is exactly point 2(a)'s "recourse to pre-trained systems or tools provided by third parties," naming which one and implicitly how it is used unmodified rather than fine-tuned. `tools_allowed` documents point 1(b)'s interactions with other

software. `fria_ref` links point 5's risk management to whatever fundamental-rights impact assessment exists. `kill_switch_url` and `retention_days` support point 3's oversight measures and connect to the Module 6 logging duties around retention. `risk_classification` is not itself an Annex IV field, but it is the fact that determines whether Annex IV applies at all, so it belongs on the card as the load-bearing flag. `contacts` supports point 1's provider identification and gives point 9's post-market monitoring plan someone to route an alert to. What the card cannot supply on its own: the rationale text behind design choices, a computational-resources figure, a dated and signed test log, a written oversight assessment, or a post-market monitoring plan. Generate the card first. Then extend it.

The GPAI Model Documentation Form, from the downstream seat

There is one more official document worth knowing about, and it belongs to someone else's obligation before it becomes useful for yours. The GPAI Model Documentation Form is the official form that providers of general-purpose AI models complete under the GPAI Code of Practice's Transparency chapter, Measure 1.1. It is not something HireSift's team fills in; it is something the provider of whatever hosted model HireSift's summarizer calls is expected to fill in.

The form flags its rows by intended recipient: some information goes to the AI Office, some to national competent authorities, and a third column is marked for downstream providers, meaning system builders like Arbeta who integrate the model rather than build it. That downstream-provider column is the part worth paying attention to from this course's seat. The rows flagged for downstream providers, things like the model's dependencies, capabilities, limitations, and energy consumption, are exactly the information you should expect to receive from your model provider, not information you have to reverse-engineer yourself.

This lands directly in your own Annex IV point 2(a): "recourse to pre-trained systems or tools provided by third parties, and how those were used, integrated or modified." If your model provider has completed the GPAI Model Documentation Form and shared the downstream-provider rows, that is the source material for your own 2(a) entry, cited rather than reconstructed from a marketing page. For BriskDesk, Solvana is in

exactly this position: BriskDesk is not high-risk and carries no Annex IV duty of its own, but when a client e-shop's due-diligence team asks what model powers the chat widget, Solvana's answer cites the downstream-provider information from its hosted-model provider's own documentation form, the same underlying artifact HireSift's Annex IV point 2(a) would cite if HireSift needed to.

Documentation as code: the assembler

The habit this module builds is a small piece of code rather than a large amount of writing: `build_skeleton(evidence_dir: Path, out_path: Path) -> Path`, in `code/governance/annex_iv_skeleton.py`. It takes a directory of evidence-pack artifacts already produced by earlier modules and emits a single markdown file structured as Annex IV's nine sections.

The mapping is direct. The assembler looks for the Module 2 classification memo, the Module 3 bias-report section, the Module 4 declared-metrics report and its evaluation-results file, the Module 5 security-testing evidence, the Module 6 logging schema, and the Module 7 oversight evidence, by their known filenames. Where an artifact exists, its content, or a summary of it, gets pulled into the matching Annex IV section. Where an artifact does not exist, for example the Article 9 risk management system, or the Article 47 declaration of conformity, the assembler writes a clearly marked TODO block naming exactly what Annex IV asks for at that point, instead of leaving the section blank or inventing filler. The assembler also generates a `ModelCard` for the system under review as part of point 1, using the same vendored `model_card.py` this chapter already covered.

```
from pathlib import Path
from governance.annex_iv_skeleton import build_skeleton

evidence_dir = Path("outputs/evidence-staging")
out_path = Path("outputs/annex_iv_skeleton.md")

build_skeleton(evidence_dir, out_path)
```

Running this against HireSift's staged evidence directory produces one markdown file with nine headed sections. Some are filled with real content pulled from Module 2 through Module 7. Some are TODO blocks naming a specific gap: the Article 9 risk management system, the Article 47 declaration, the Article 72(3) post-market monitoring plan Module 9 will build. Counting filled sections against TODO sections after a run is itself a useful compliance metric: it tells you, at a glance, how much of Annex IV your existing engineering practice already covers, and exactly what is left.

The habit worth keeping past this module is running the assembler again every time an earlier module's output changes, rather than treating `annex_iv_skeleton.md` as a document someone edits by hand. Hand edits are exactly what "kept up-to date" in Article 11(1) is designed to prevent from being the only mechanism; a document assembled by code from source artifacts is a document that cannot silently drift from what those artifacts actually say.

Keeping documentation current

Article 11(1)'s second duty, "kept up-to date," and Annex IV point 6's lifecycle-changes description are really one requirement told twice. A handful of events should trigger a documentation rebuild every time: a model swap (the hosted LLM behind the summarizer changes version), a prompt-template edit, a new deployment context (a client integrates HireSift into a different ATS workflow than before), or a retrain of the ranking model on a new slice of historical data.

Module 6's system-change events are already the changelog point 6 asks for; they exist whether or not anyone remembers to update a documentation file by hand. The remaining piece, which Module 9 builds, is wiring the assembler into the same pipeline that ships a release, so that a change event does not just get logged, it triggers a fresh run of `build_skeleton` and a fresh `annex_iv_skeleton.md`. Documentation that regenerates on every release cannot rot the way documentation edited by hand, occasionally, under deadline pressure, reliably does.

Who reads this

Writing for an abstract “compliance” audience produces vague documentation. Writing for the specific readers Annex IV documentation actually has produces useful documentation. Four readers matter in practice: notified bodies, for systems that go through the Annex VII third-party conformity-assessment route; market surveillance authorities, who can request the technical documentation under their Article 74 powers whether or not a notified body was ever involved; your own deployers, since the instructions for use are literally Annex IV point 1 and a deployer who cannot understand HireSift’s declared limitations cannot use it responsibly; and a client’s due-diligence team, who will ask a version of these questions long before any regulator does, because procurement now routinely asks vendors to demonstrate exactly this kind of evidence. Write for the hostile-but-competent reader in each of these seats: someone who knows the domain, does not already trust you, and is going to check the parts that matter.

Worksheet: your own point-by-point self-audit

Pick one system you build, maintain, or integrate. Work through Annex IV’s nine points and, for each one, note whether an artifact already exists that would satisfy it, or whether it would be an honest TODO today. Do not round up; a folder of scattered notes is not the same as a datasheet, and a green CI badge with no date or name attached is not the same as a signed test log.

Then write point 1, the general description, in full: intended purpose, provider name and version, how the system interacts with other hardware or software, the forms in which it is placed on the market, the hardware it runs on, whether it is a component of a physical product, a basic description of the deployer’s interface, and instructions for use. Write it as if a notified body will read it next week. A sample answer for HireSift follows in the appendix, but write yours first; the exercise is in the choices you make before you see someone else’s.

Appendix: HireSift, Annex IV point 1, sample answer

Illustrative worked example for the fictional HireSift system (Arbeta GmbH, see CAST.md). Specific figures such as the version number are invented for the exercise,

not real Arbeta data.

(a) Intended purpose, provider, version. HireSift's intended purpose is to parse candidate CVs submitted through a client's applicant-tracking system, extract structured features, score and rank candidates against a job profile, filter the bottom of the ranked pool before a recruiter sees it, and generate a short summary of each shortlisted candidate. The provider is Arbeta GmbH, Vienna. This documentation covers version 3.1, whose relation to the previous version is a replacement of the linear scoring model with a gradient-boosted ranker trained on the same historical outcome data, with no change to the summarizer.

(b) Interactions with hardware or software. HireSift is a module integrated into a client's applicant-tracking system through a documented API; it consumes parsed CV data the ATS provides and returns scores, ranks, and shortlist summaries back to that system. It calls a hosted large language model through an external API for the summarization step (see point 2(a) for the pre-trained-tool detail); it does not interact with any other AI system directly.

(c) Relevant software or firmware versions and update requirements. HireSift is delivered as a versioned API integration; clients must be running an ATS release compatible with the current HireSift API contract, and Arbeta issues a compatibility note with each HireSift release.

(d) Forms of placing on market. HireSift is placed on the market exclusively as a licensed API integration embedded within client applicant-tracking systems; it is not distributed as a standalone download or a hardware-embedded package.

(e) Hardware. HireSift's scoring and ranking components run on Arbeta's own cloud infrastructure; no specific end-user hardware is required beyond a standard workstation running the client's ATS.

(f) Photographs or illustrations. Not applicable; HireSift is not a component of a physical product.

(g) Basic description of the deployer user interface. Recruiters interact with HireSift through a ranking dashboard embedded in the ATS: a sorted candidate list

with a numeric score, a short rationale summary per candidate, and a flag showing which candidates were filtered out before reaching the list, with the reason.

(h) Instructions for use. The instructions state what the score represents (a relative ranking signal against the stated job profile, not a hiring decision), what it does not represent (no claim of a candidate's overall suitability outside the profile it was scored against), the declared accuracy figures and subgroup breakdown from point 3, and an explicit instruction that a human recruiter must review the shortlist and rationale before contacting any candidate, consistent with the Article 14 oversight measures documented under point 2(e).

CHAPTER 9

Compliance in the Pipeline: Articles 72 and 73

Module 4 built an evaluation machine that re-runs the declared metrics on demand. The record-keeping work built a log that survives past the demo. Module 8 turned all of it into a documentation folder with a defensible structure. Articles 72 and 73 are the pair of duties that assume all three keep running after the conformity assessment is filed and the system is out on the market. Nothing in this chapter is new engineering. It is the same eval suite, the same logs, the same documentation, wired to a calendar instead of a launch date.

Article 72(2) states the idea in a sentence that reads like a pipeline description written by a lawyer: the monitoring system shall "actively and systematically collect, document and analyse" data on performance "throughout their lifetime." A release is a snapshot. The obligation this chapter covers is about the movie: the system as it behaves in month three, month twelve, and month thirty-six after the demo that got everyone excited.

Article 72: post-market monitoring as a documented system

Here is the text, unmodified, from Chapter IX, Section 1 of the enacted Act.

THE ACT SAYS · ARTICLE 72(1)-(3)

1. Providers shall establish and document a post-market monitoring system in a manner that is proportionate to the nature of the AI technologies and the risks of the high-risk AI system.
2. The post-market monitoring system shall actively and systematically collect, document and analyse relevant data which may be provided by deployers or which may be collected through other sources on the performance of high-risk AI systems throughout their lifetime, and which allow the provider to evaluate the continuous compliance of AI systems with the requirements set out in Chapter III, Section 2. Where relevant, post-market monitoring shall include an analysis of the interaction with other AI systems. This obligation shall not cover sensitive operational data of deployers which are law-enforcement authorities.
3. The post-market monitoring system shall be based on a post-market monitoring plan. The post-market monitoring plan shall be part of the technical documentation referred to in Annex IV. The Commission shall adopt an implementing act laying down detailed provisions establishing a template for the post-market monitoring plan and the list of elements to be included in the plan by 2 February 2026.

Three paragraphs, three jobs. Paragraph 1 is the existence duty: build a monitoring system, write it down, and size it to the system's actual technology and risk, not to a generic template copied from a bigger company's compliance binder. Paragraph 2 is the behaviour duty, and it is the one worth reading twice. "Actively and systematically" rules out a monitoring system that consists of waiting for a deployer to complain. The data can come from deployers or from other sources the provider collects on its own, analysed against a specific target: continuous compliance with the Chapter III, Section 2 requirements, the same accuracy, robustness, data-governance, oversight and logging duties Modules 3 through 7 built evidence for. Where the system talks to other AI systems, the analysis has to cover that interaction too. One carve-out is narrow: sensitive operational data belonging to law-enforcement deployers is out of scope, relevant to systems like those under Annex III point 6 but not to HireSift or BriskDesk.

Paragraph 3 does two things at once. It says the monitoring system rests on a written plan, and it places that plan inside the technical documentation from Annex IV, specifically at point 9 of that Annex's list. The monitoring plan is not a side document living in an operations wiki; it is part of the same Annex IV file Module 8 built, filed alongside the architecture description, the risk-management summary, and the declared metrics. A provider who treats post-market monitoring as an ops-team concern separate from the compliance folder has already split something the Act keeps together.

> AS OF MID-2026
>
> Paragraph 3 also names a deadline that is easy to misread. The Commission was due to adopt an implementing act with a template for the plan, and a list of elements the plan must include, by 2 February 2026. As of this writing in mid-2026, no adopted template of this kind appears in the reference material this course draws from. That is a statement about what has not been confirmed, not a claim that the template does or does not exist by the time you are reading this. Check the current status before you file anything against it. If a template has since been adopted, use it; if it has not, a documented, proportionate monitoring plan built along the lines this chapter describes is still what paragraphs 1 and 2 require in the meantime, template or no template.

Paragraph 4, not quoted above, lets providers of high-risk systems under Annex I Section A product law, and financial institutions covered by point 5 of Annex III, fold the Article 72 elements into a monitoring system they already run under existing sector rules, rather than standing up a parallel one. None of the three cast systems in this course sit in that lane, but the option is worth knowing if your own system does.

The monitoring plan, made concrete: a signals table

"Actively and systematically" is a legal phrase. In engineering terms it means a table with four columns: what you watch, where the data comes from, what threshold trips a response, and what the response is. The table below is HireSift's monitoring plan, and

every row reuses an artifact this course already built. That reuse is the point: a monitoring plan invented from scratch at the post-market stage is exactly the kind of paper compliance the Act's proportionality language is trying to head off.

SIGNAL	SOURCE	THRESHOLD	ACTION
Declared-metric drift	Module 4's evaluation suite, re-run on a schedule	Precision, recall, or subgroup true-positive rate drops below the level declared under Article 15(3)	Re-run the full suite; quarantine the release if the drop holds; update the declaration if the shift is genuine and sustained
Input distribution shift	The record-keeping log built for Article 12	Feature distributions on live CVs diverge from the last-validated population past an agreed threshold	Flag for review; pull a fresh sample; re-validate the declared metrics against it
Subgroup gap movement	The bias-examination slices from Module 3, re-applied in Module 4's suite	The true-positive-rate gap across a protected characteristic widens beyond the band examined at launch	Re-open the bias examination as a monitoring-triggered re-assessment, not a one-off anomaly
Override rate spike	The human-oversight measurement from Module 7	Reviewers overriding HireSift's ranking at a rate meaningfully above baseline	Investigate whether the model degraded or the reviewers changed; either answer is monitoring data
Tool-abuse attempts	The security log from Module 5	A spike in injection or tool-misuse attempts against the scoring pipeline	Feed into the next adversarial-testing sweep as performance data on the system's real operating environment
Deployer complaints	An intake channel Arbeta runs for its employer clients	Any complaint alleging a discriminatory or clearly wrong outcome	Log immediately as a candidate serious incident and start the

SIGNAL	SOURCE	THRESHOLD	ACTION
			Article 73 classification below

Read left to right, the table does what Article 72(2) asks: collect data (columns one and two), analyse it against a threshold (column three), and produce an action that confirms continuous compliance or starts a corrective loop (column four). None of the six rows required new instrumentation, only a decision to keep instrumentation already built for Modules 3 through 7 running after the demo ends.

Article 73: reporting serious incidents

Section 2 of Chapter IX is shorter and sharper than Section 1, because it deals with the moment monitoring catches something that already went wrong for a real person. Here is the operative text.

THE ACT SAYS · ARTICLE 73(1)-(6)

1. Providers of high-risk AI systems placed on the Union market shall report any serious incident to the market surveillance authorities of the Member States where that incident occurred.
2. The report referred to in paragraph 1 shall be made immediately after the provider has established a causal link between the AI system and the serious incident or the reasonable likelihood of such a link, and, in any event, not later than 15 days after the provider or, where applicable, the deployer, becomes aware of the serious incident. The period for the reporting referred to in the first subparagraph shall take account of the severity of the serious incident.
3. Notwithstanding paragraph 2 of this Article, in the event of a widespread infringement or a serious incident as defined in Article 3, point (49)(b), the report referred to in paragraph 1 of this Article shall be provided immediately, and not later than two days after the provider or, where applicable, the deployer becomes aware of that incident.
4. Notwithstanding paragraph 2, in the event of the death of a person, the report shall be provided immediately after the provider or the deployer has established, or as soon as it suspects, a causal relationship between the high-risk AI system and the serious incident, but not later than 10 days after the date on which the provider or, where applicable, the deployer becomes aware of the serious incident.
5. Where necessary to ensure timely reporting, the provider or, where applicable, the deployer, may submit an initial report that is incomplete, followed by a complete report.

> 6. Following the reporting of a serious incident pursuant to paragraph 1, the provider shall, without delay, perform the necessary investigations in relation to the serious incident and the AI system concerned. This shall include a risk assessment of the incident, and corrective action. The provider shall cooperate with the competent authorities, and where relevant with the notified body concerned, during the investigations referred to in the first subparagraph, and shall not perform any investigation which involves altering the AI system concerned in a way which may affect any subsequent evaluation of the causes of the incident, prior to informing the competent authorities of such action.

Paragraph 1 answers “where”: the market surveillance authority of the Member State where the incident occurred, not a central EU body and not the authority of wherever the provider happens to be established. Paragraphs 2 through 4 are three clocks layered over the same duty, all sharing the same starting gun.

What counts, and the three clocks

Article 3, point (49) is where “serious incident” gets defined, and Article 73 leans on two of its limbs by name. Article 73(3) triggers the fastest clock for “a serious incident as defined in Article 3, point (49)(b)”, which covers serious and irreversible disruption of the management or operation of critical infrastructure. Article 73(7), (9) and (10) each refer to “Article 3, point (49)(c)”, which covers infringement of obligations under Union law intended to protect fundamental rights. Alongside those two named limbs, the definition also covers death or serious harm to a person’s health, and serious harm to property or the environment. Four categories, in plain terms: someone died or was seriously harmed; critical infrastructure broke in a way that cannot be quietly reversed; a fundamental right was infringed; or property or the environment took serious damage.

The clock that applies depends on which category you are in, and every clock starts from the same moment: awareness, not confirmation.

INCIDENT CLASS	DEADLINE	STARTS FROM
Widespread infringement, or a serious incident under Article 3, point (49)(b) (critical-infrastructure disruption)	Immediately, not later than two days	The provider, or the deployer, becoming aware
Death of a person	Immediately once causality is established or suspected, not later than 10 days	The provider, or the deployer, becoming aware
Standard serious incident (the general case, including Article 3, point (49)(c) fundamental-rights incidents that are not also widespread)	Immediately once a causal link, or a reasonable likelihood of one, is established, not later than 15 days	The provider, or the deployer, becoming aware

WATCH OUT

Two details trip people up. First, the clock is anchored to awareness, not to when a formal investigation confirms causation. Article 73(2) sends the report out "immediately after the provider has established a causal link… or the reasonable likelihood of such a link", with the day count as the outer bound, not the target; reasonable likelihood starts the clock, full certainty is never a precondition for filing. Second, severity shortens the standard window rather than lengthening it. Article 73(2)'s "the period for the reporting… shall take account of the severity of the serious incident" is the general principle the two-day and ten-day carve-outs in paragraphs 3 and 4 turn into hard numbers.

Filing early, filing complete: the initial incomplete report

Article 73(5) exists because real incidents rarely arrive with a tidy, complete picture on day one. It allows "an initial report that is incomplete, followed by a complete report." Read against the deadlines above, this reframes what "reporting" means in practice: the duty is to get something to the authority inside the clock, not to have finished the investigation inside the clock. A provider who spends thirteen of their fifteen days trying to build a perfect first report, rather than filing an honest partial one on day two and completing it later, has misread paragraph 5.

The evidence-preservation rule: Article 73(6)

Once the report is filed, paragraph 6 imposes two duties in the same breath: investigate without delay, including a risk assessment and corrective action, and cooperate with the competent authorities and any notified body involved. Buried in the same paragraph is the rule every on-call engineer needs memorised before the first real incident: the provider "shall not perform any investigation which involves altering the AI system concerned in a way which may affect any subsequent evaluation of the causes of the incident, prior to informing the competent authorities of such action."

In practice this means quarantine before you patch. The instinct in a live incident is to fix the bug the moment you find it. Article 73(6) asks for the opposite sequence when the fix would alter the system in a way that could affect how the authority later evaluates what happened: freeze the affected version, preserve the logs and the model artifact exactly as they were at the moment of the incident, tell the authority what you are about to do, and only then make the change. Corrective action is required; corrective action performed as a silent hotfix-and-overwrite, before the authority has been told, is the specific failure mode this paragraph exists to prevent.

The official artifacts, and their honest status

Four things are worth knowing about the paperwork side of Article 73, and precision matters here because two of the four are still in motion.

The draft Article 73 guidance. Article 73(7) required the Commission to issue guidance to facilitate compliance with the reporting obligation, and set 2 August 2025 as the issue date. A draft exists in the reference material this course draws from, dated September 2025. As of mid-2026 it is a draft: treat its structure as informative and its content as not yet finalised.

The draft Article 73 report template. Alongside the guidance, a draft template for the high-risk incident report exists, also dated September 2025, structured into five sections: administrative information, AI system information, incident information, provider analysis, and general comments. This is the shape the worked example later in

this chapter follows, and a reasonable structure to build your own runbook around even while it remains a draft.

Where reports go. Article 73(1) sends the report to the market surveillance authority of the Member State where the incident occurred, and Article 73(8) gives that authority seven days from receipt to take appropriate action under the market-surveillance framework.

A separate track for GPAI providers. A distinct incident-report template tied to Article 55 and systemic-risk general-purpose AI models exists, dated November 2025. It binds providers of general-purpose AI models with systemic risk, a different obligation on a different chapter, and is not the template a high-risk system provider like Arbeta files against. The two tracks are easy to conflate because both concern "AI incidents"; mixing them up would send the wrong document to the wrong authority.

The runbook pattern

None of the above is useful at two in the morning unless it has already been turned into a document your on-call engineer can fill in without re-reading Article 73 first. The course's vendored governance code includes exactly that: an `Incident` dataclass and a `render()` function that turns a filled-in incident into a runbook with real clocks computed from the moment of awareness.

```
@dataclass
class Incident:
    incident_id: str
    system_id: str
    version: str
    awareness_at: datetime     # the moment the provider became aware
    severity: str               # widespread | death | standard
    description: str
    immediate_actions: list[str] = field(default_factory=list)
    contacts: dict[str, str] = field(default_factory=dict)
    fria_ref: str = ""

_DEADLINES = {
    "widespread": timedelta(hours=48),
```

```
    "death": timedelta(days=10),
    "standard": timedelta(days=15),
}
```

The three entries in `_DEADLINES` are the same three clocks from the table earlier in this chapter, expressed as code instead of prose. `render()` takes an `Incident`, adds the right deadline to `awareness_at`, and writes out a runbook with the reporting deadline stated in plain text, alongside the description, the immediate actions already taken, the sign-off contacts, and a reporting-channel checklist pointing at the runbook itself, the relevant log slice, the model card and FRIA at the deployed version, and the most recent red-team report for the system. The generated file goes in `artifacts/` and is meant to be regenerated per incident, not written once and reused with the details swapped by hand.

The discipline this encodes is worth stating plainly: decide the paging tree, the deadline table, and the report skeleton before the first incident, not during it. An incident that starts with someone searching for “who do we even call” has already lost hours off a two-day clock.

Evidence regeneration as CI

Everything in this chapter so far describes what happens when monitoring catches a signal or an incident occurs. The other half of Article 72’s “actively and systematically” is what happens when nothing has gone wrong yet: the evidence has to keep regenerating anyway, on a schedule, without anyone needing to remember to run it.

The mechanism is a small continuous-integration job that re-executes the same evaluation notebook Module 4 built, on every release and on a recurring schedule, and fails the build if the result has quietly gotten worse. Step by step, the job triggers on every push and also on a schedule, so drift gets caught in a quiet week with no releases; sets up a clean Python environment with the small set of packages the notebook needs; re-executes the Module 4 notebook headlessly, exactly as it runs when a student steps through it by hand, producing a fresh `eval_results.json` and `declared_metrics.md` from whatever data is current, not a cached result committed

months ago; loads that fresh result and compares the overall pass rate against a threshold, failing the build if it dropped; and uploads the notebook's output directory as a build artifact with a retention period, so the evidence outlives the job that produced it.

A condensed version of the workflow file looks like this:

```
name: evidence

on:
  push:
  workflow_dispatch:
  schedule:
    - cron: "0 6 * * 1"   # weekly, Monday morning

jobs:
  regenerate-evidence:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
        with:
          python-version: "3.12"
      - run: pip install nbconvert jupyter pandas matplotlib
      - run: >
          jupyter nbconvert --to notebook --execute
          notebooks/m04_art15_evidence.ipynb
          --output executed_m04.ipynb
      - name: threshold gate
        run: python - <<'PY'
          # load notebooks/outputs/eval_results.json, read
          # overall_pass_rate, exit non-zero below EVAL_THRESHOLD
          PY
      - uses: actions/upload-artifact@v4
        with:
          name: eval-evidence
          path: notebooks/outputs/
          retention-days: 190
```

The threshold-gate step is deliberately plain Python, inlined in the workflow itself (the full version lives in `exercises/m09-ci/evidence.yml`): load

`notebooks/outputs/eval_results.json`, read `overall_pass_rate`, compare it against the `EVAL_THRESHOLD` environment variable, and exit non-zero if the run fell short, the same pass/fail logic a student runs by hand in the notebook, just moved into a place where nobody has to remember to run it.

The retention period deserves an honest note. Neither Article 72 nor Article 73 sets a specific retention period for CI-generated evidence artifacts; that is a separate concern from the multi-year retention duties the Act places on technical documentation more broadly. The 190 days chosen here is an engineering margin, not a legal citation: six months is a common floor teams use for evidence that might get pulled into an incident review, and the extra ten days give slack for a late-discovered issue before the artifact expires. Set your own number deliberately, and do not mistake it for something the Act requires.

The principle underneath all of this: evidence is a build artifact with a retention setting, not a folder on someone's laptop updated the week before an audit.

Sectoral overlaps: when reporting narrows

Article 73(9) and 73(10) both narrow the reporting duty for specific sectors. The honest version is that none of Arbeta's, Solvana's, or Norrfelt's systems currently sit in either narrowed lane, but the narrowing matters for other providers reading this. Under Article 73(9), where a high-risk system under Annex III is placed on the market by a provider already subject to Union legislation with equivalent reporting obligations, the notification duty is limited to incidents under Article 3, point (49)(c), the fundamental-rights limb only; the other three categories are assumed covered by the equivalent regime already in force. Article 73(10) narrows the same way for high-risk systems that are themselves medical devices, or safety components of medical devices, covered by the EU Medical Devices Regulation or In Vitro Diagnostic Regulation: reporting is limited to Article 3, point (49)(c) incidents, sent to the national competent authority the Member State designates for that purpose rather than the general market surveillance authority.

A device manufacturer or a system covered by an equivalent EU reporting regime does not get to skip Article 73 entirely, it gets a narrower version of it, one that still needs a monitoring plan built to catch the fundamental-rights category specifically.

Worked in full: the HireSift works-council dry run

A works council at one of Arbeta's employer clients files a formal complaint: HireSift's shortlisting has systematically filtered out older applicants over the past hiring cycle. This is a fundamental-rights incident candidate under Article 3, point (49)(c), and it is exactly the kind of complaint the intake-channel row of the monitoring table earlier in this chapter exists to catch. Here is the drill Arbeta runs, end to end.

Step one: log the awareness timestamp. The moment the complaint reaches Arbeta's intake channel is the moment the clock starts, regardless of how long the investigation that follows takes. This timestamp goes into the incident record before anything else happens.

Step two: classify severity and pick the clock. This is not a widespread infringement across multiple Member States on the facts as they currently stand, it is not a critical-infrastructure disruption, and nobody died. It is a fundamental-rights incident under Article 3, point (49)(c), which falls under the standard clock: 15 days from awareness, per Article 73(2). Arbeta logs "standard, 15-day clock" against the awareness timestamp from step one.

Step three: draft the initial incomplete report. Following the draft template's five-section structure, Arbeta fills in what it can within the window rather than waiting for a complete investigation:

- *Section 1, administrative information*: Arbeta GmbH as provider, the affected employer as deployer, HireSift's system identifier and deployed version, the works council's complaint reference.
- *Section 2, AI system information*: HireSift's intended purpose (CV screening and ranking for recruitment), its Annex III point 4(a) classification, and the deployed model version at the time of the alleged filtering.

- *Section 3, incident information*: the awareness timestamp, a description of the alleged filtering pattern as reported by the works council, and an explicit statement that this is an initial report under Article 73(5) with the investigation ongoing.
- *Section 4, provider analysis*: whatever Arbeta can state at this early stage, flagged as preliminary, pending the evidence pulled in step four.
- *Section 5, general comments*: the corrective-action process that has been opened, and a note that a complete report will follow.

Step four: pull the evidence. Arbeta pulls exactly the artifacts this course built: the age-proxy analysis from Module 3's bias examination (address, employment-gap length, and graduation-year fields have all shown up as age proxies in similar systems), the current subgroup true-positive-rate gap from Module 4's declared-metrics suite re-run against the affected employer's recent applicant pool, and the relevant slice of the Article 12 record-keeping log covering the disputed scoring decisions.

Step five: open the corrective-action log, and write the 73(6) note. Before any model change is made, Arbeta records a formal note: the currently deployed version is quarantined and will not be silently modified; if a fix to the scoring logic is warranted once the investigation concludes, the competent authority will be informed of the intended change before it is made, per Article 73(6). This note exists so that a well-meaning engineer patching the ranking weights on a Friday afternoon does not destroy the evidence the authority would need to evaluate what caused the filtering pattern.

Five steps, and every one reused an artifact this course already built or filled in a template section this chapter already described. The incident dry run is fast precisely because nothing in it is improvised.

Worksheet: BriskDesk's monitoring-plan signals table

BriskDesk is not high-risk. Solvana carries no Article 72 obligation for it. Build the signals table anyway, the same way Module 4's chapter had Norrfelt declare metrics for VibraSense without a legal duty to do so, because a support chatbot nobody is watching in production is a bad idea independent of legal tier.

Using what you know about BriskDesk (a RAG-based support chatbot over each client's product catalog and help-center articles, with tool calls to the client's order API, and the evaluation pipeline built in Module 4), write your own version of the signals table from earlier in this chapter. For each row, name:

1. The signal: what specifically would tell Solvana something has changed for the worse.
2. The source: which artifact BriskDesk already produces (or should produce) that carries this signal.
3. The threshold: a concrete condition that would trigger a response, not just "if it seems off."
4. The action: what Solvana actually does when the threshold trips.

Aim for at least four rows before checking the appendix.

Appendix: sample answer

SIGNAL	SOURCE	THRESHOLD	ACTION
Grounding-rate drop	Module 4's evaluation pipeline, re-run on schedule	Faithfulness or grounding-rate metric falls below the declared level	Re-run against the latest RAG index; check the index for staleness before suspecting the model
Escalation-precision drift	The pipeline's escalation-category results	Correct-escalation rate drops meaningfully below baseline	Sample recent escalations by hand; check whether the catalog lagged a product-line change
Upstream model swap	Hosted LLM provider's API version tag, checked against the pinned regression suite	Provider ships a new model version behind the same tag	Hold the upgrade; run the pinned regression suite; adopt only if it passes

SIGNAL	SOURCE	THRESHOLD	ACTION
Prompt-injection or tool-misuse attempts	The security log from Module 5	Spike in attempts to manipulate the RAG index or misuse order/return tools	Feed into the next security sweep; tighten input handling if the pattern is new
Client complaints	Solvana's support intake per e-shop client	Any complaint of a wrong order status, invented policy, or unauthorised return	Log as an internal incident; investigate, correct, re-test before the next release

None of these five rows is required by the Act for BriskDesk. All five are the same discipline Article 72 asks of HireSift, applied because a chatbot that quietly gets worse after launch is a business problem whether or not a regulator is watching for it.

The arc, closed

Release is a snapshot. Monitoring, incident reporting, and evidence that regenerates on a schedule are how a system stays compliant after the snapshot was taken. That is the whole shift this chapter asks for: stop treating the eval suite, the logs, and the documentation folder as things you finished, and start treating them as things that run.

CHAPTER 10

Capstone: The Evidence Pack

Module 1 made a promise on slide 11: an `evidence-pack/` directory, one folder per obligation, that an auditor, a client's due-diligence team, or your own future self could open and read. Nine modules later, every piece of that promise already exists, scattered across notebook outputs, exercise folders, and worksheet answers. This module does not add a new article or a new legal claim. It assembles what is already there, checks it honestly, and hands you the same process to run on a system of your own.

That framing matters more than it sounds like it should. Assembly does real work rather than a formality tacked on at the end. A folder of correct artifacts that nobody has walked end to end, checking that the pieces still agree with each other, is raw material waiting to become evidence. This chapter covers the walk that turns one into the other.

The tree, revisited

Here is the same tree from Module 1's slide 11, with every line annotated by the module that fills it and the article it answers to.

```
evidence-pack/
├── 01_classification_memo.md        # Module 2 - Article 6(3)-(5), Article 6(4) documen
├── 02_data_governance/
│   ├── dataset_datasheet.md         # Module 3 - Article 10(2)-(5)
│   └── bias_report_section.md       # Module 3 - Article 10, bias examination
├── 03_evaluation/
│   ├── declared_metrics.md          # Module 4 - Article 15(3)
│   ├── eval_results.json            # Module 4 - Article 15(1)-(4)
│   └── security_tests.md            # Module 5 - Article 15(5)
├── 04_logging_schema.md             # Module 6 - Article 12, Article 19 retention
├── 05_oversight_measurement.md      # Module 7 - Article 14
├── 06_technical_documentation/      # Module 8 - Article 11, Annex IV
│   ├── annex_iv_skeleton.md
│   └── model_card_hiresift.json
└── 07_monitoring_and_incidents.md   # Module 9 - Article 72, Article 73
```

Two small honesty notes on the mapping itself. Module 1's original preview named `bias_report.html`, a geobias-style rendered artifact; in practice this course produces `bias_report_section.md`, the written section Module 3 builds from that evaluation, and the assembler normalizes the one into the other rather than pretending both exist. Module 4's `declared_metrics.md` was introduced separately from `eval_results.json` in that chapter's own worksheet; both land inside `03_evaluation/`, because Article 15(3)'s declared metrics and Article 15(1)-(4)'s evaluation results are the same evidence read two ways, a declaration and the run that backs it.

`capstone/assemble_evidence_pack.py` is what actually builds this tree. For each target path it holds a short priority list of candidate sources: your own notebook outputs first, the course's committed sample copies as a fallback if a module was skipped. It copies whatever exists into `capstone/evidence-pack/`, and writes `INDEX.md`, a manifest naming, per file, which module produced it and which article it answers, plus a completeness table marking each artifact present, filled from a fallback, or missing. A missing artifact is never silently skipped; the manifest names the exact notebook to run to fill the gap. The table below is the same mapping, read as a status check rather than a file tree.

ARTIFACT	ARTICLE	MODULE	STATUS IF SKIPPED
Classification memo	Art 6(3)-(5)	M02	course fallback: HireSift memo
Datasheet + bias section	Art 10	M03	course fallback: geobias sample
Declared metrics + eval results	Art 15(1)-(4)	M04	course fallback: sample run
Security tests	Art 15(5)	M05	course fallback: sample run
Logging schema	Art 12, Art 19	M06	course fallback: HireSift schema
Oversight measurement	Art 14	M07	course fallback: sample session

ARTIFACT	ARTICLE	MODULE	STATUS IF SKIPPED
Technical documentation	Art 11, Annex IV	M08	course fallback: skeleton + model card
Monitoring and incidents	Art 72, Art 73	M09	course fallback: signals table + dry run

A pack built entirely from course fallbacks tells you nothing about your own system. It is useful exactly once, as a worked example of the shape a finished pack takes, before you replace every row with your own work in the assignment later in this chapter.

The rubric: four checks, and what failing each one looks like

`capstone/README.md` states the rubric this module has been building toward since Module 1: for every artifact, check whether it **exists**, is **grounded**, is **honest**, and is **current**. Passing means all four, for every artifact, not most of them for most artifacts.

Exists. The file is present and contains the substance the artifact is supposed to carry, not a heading and a placeholder. *Pass*: `03_evaluation/eval_results.json` holds the actual output of a re-run of `notebooks/m04_art15_evidence.ipynb`, with real numbers attached to named metrics. *Fail*: an oversight-measurement report that opens with the right section headings from Module 7's template but never fills in a single number under any of them. A document with the right shape and no content inside it only satisfies "filename exists," which the rubric does not count as a pass.

Grounded. The numbers in the artifact trace to a specific run, over a named test population, and that run is the one currently sitting in the pack, not an older or different one. *Pass*: `declared_metrics.md` names precision at the shortlist cutoff and the subgroup true-positive-rate gap, states the applicant-pool date range the numbers were measured against, and those numbers match what `eval_results.json` actually contains in this same pack. *Fail*: a metrics section that states an accuracy figure with no test population named at all, so there is no way to tell whether the number came from a curated benchmark, last year's training data, or this year's live traffic. Module 4 made

this exact point about declared metrics: a number without a named population is a number that hides the question an auditor asks first.

Honest. The artifact states what remains wrong, not only what has been fixed. *Pass*: `bias_report_section.md` reports the subgroup gap that persists after Module 3's mitigation work, names it as a residual risk still open, and does not round it down to zero. *Fail*: a bias section that reports zero remaining disparity across every protected characteristic examined. Module 3's own bias taxonomy makes the honest version of this section hard to reach by accident; a zero-gap claim across the board is a documentation failure before it is anything else, because historical hiring data of the kind HireSift trains on essentially never produces a clean zero.

Current. The artifact is dated, and it names the event or the interval that would trigger a rebuild. *Pass*: `04_logging_schema.md` carries a date and states plainly that a schema-version bump in Module 6's event taxonomy triggers a rewrite of this document, not a silent edit. *Fail*: a classification memo that carries a date from when the system was first built, three years back, with no review trigger named anywhere in it. Such a memo may have been correct on the day it was written, but nothing in it lets anyone know when that stopped being true.

> WATCH OUT
>
> Four checks, and the failure mode in each case is the same shape: a document that looks finished because it has the right title and the right section headings, while the actual work the section exists to do never happened.

The auditor's walkthrough, and the two questions

An auditor, or a client's due-diligence reviewer, does not read this pack top to bottom in file order. They read it in dependency order, and the order matters because later files are conditional on earlier ones.

Classification first. Everything else in the pack only makes sense once `01_classification_memo.md` has answered which lane the system sits in. A high-risk

verdict is what makes `03_evaluation/`, `05_oversight_measurement.md`, and the rest legally owed rather than merely good practice. Module 2 built this memo precisely because it decides the workload for every module that follows; an auditor reads it first for the same reason the course built it first.

Documentation as the hub.
`06_technical_documentation/annex_iv_skeleton.md` is read second, not because it is the most important artifact on its own, but because Module 8 built it to point into every other artifact by section: point 2's testing and cybersecurity sub-points into the eval and security evidence, point 2's data requirements into the datasets described in Module 3, point 9 into the post-market monitoring plan that closes into `07_monitoring_and_incidents.md`. An auditor follows those pointers outward from the skeleton rather than reading seven unrelated files and guessing how they connect.

Whatever artifact the auditor lands on, it has to survive two questions, the same two questions in every case:

"How do you know?" Point to the specific, dated run that produced the number or the claim in front of you. "We tested it" fails to answer the question. "Precision at cutoff, 0.78, measured against the March applicant pool, `eval_results.json`, run on this date" answers it.

"What happens when it changes?" Name the trigger and the cadence. A model swap, a retrain, a schema-version bump, a quarterly calibration check, a drift alarm from Module 9's monitoring loop; whichever one applies, the artifact has to say which one and what happens when it fires.

An artifact that survives both questions for every claim it makes has passed the rubric in substance, whatever the four checkboxes say on paper.

What this pack deliberately does not include

Honesty about a pack's contents means being equally clear about its edges. Four things this course has referenced but never built sit outside `evidence-pack/` entirely, and the pack says so rather than staying silent about them.

The Article 9 risk management system. Module 8's Annex IV skeleton leaves this as a named TODO block wherever the assembler cannot find it, rather than a blank section or invented filler. A risk management system spanning the full product lifecycle is a governance program, not a single artifact a notebook produces.

The Article 47 declaration of conformity and **Article 49 registration** in the EU database. Both depend on completing the conformity assessment procedure this course has not run, whether that is self-assessment against Annex VI or a notified body's involvement under Annex VII. Signing a declaration or registering a system ahead of that assessment would be the opposite of honest.

An Article 17 quality management system. Nothing in this course builds one. A QMS is an organization-wide operating system for how a provider builds, tests, and maintains AI systems across every product it ships, not a per-system evidence folder.

All four live in the same place: conformity-assessment territory, alongside the introduction course's legal depth and, for anything with real stakes, a lawyer who can weigh the specific facts. This pack functions as the engineering evidence layer underneath a conformity file; the conformity file itself, with its risk management system, declaration, registration, and QMS, still has to be built separately. Naming that gap plainly, inside the pack itself, is what keeps the rest of the pack credible. A pack that quietly implied it covered the whole compliance story would fail its own honesty check on the first page.

Your own system: the assignment

The M01 worksheet asked you to sort your own systems into three lanes. This is the same question, run one more time, all the way to an assembled pack.

1. **Pick one system you actually build, maintain, or integrate**, or return to one of the three from the Module 1 worksheet.
2. **Classify it, using Module 2's memo template.** Run the Article 6(3) derogation filter honestly; check the draft classification guidelines for the closest worked example if the case is ambiguous.

3. **Let the classification decide the workload.** A high-risk verdict owes the full pack: every module's evidence, in full. A limited-risk verdict owes the Article 50 transparency statement plus whichever good-practice subset of the pack you choose to keep anyway (Module 4 and Module 9 both showed BriskDesk keeping declared metrics and a monitoring signals table with no legal duty to do either). An out-of-scope verdict owes the Article 6(4) memo itself, plus whatever good engineering you decide is worth doing regardless, the way Norrfelt keeps VibraSense's declared-metrics section on file although nothing in Annex III currently reaches it.
4. **Rerun whichever notebooks apply**, over your own data, not the course's cached HireSift or BriskDesk runs.
5. **Point `assemble_evidence_pack.py`'s source list at your own output paths**, and run it.
6. **Walk your own `INDEX.md`** the way the auditor walkthrough above describes: classification first, documentation as the hub, every artifact checked against exists, grounded, honest, current.
7. **Ask both auditor questions of your own pack before anyone else does.** If an artifact cannot answer "how do you know" or "what happens when it changes," that is the artifact to fix before this assignment counts as done.

Keeping it true

A pack assembled once and filed away starts decaying the day it is written. Keeping it true is the discipline Module 9 already built the machinery for: a change event (a model swap, a retrain, a new deployment context, a drift alarm crossing threshold) triggers a rebuild, not a calendar reminder someone eventually ignores. That loop, already wired into CI in Module 9's exercise, is what makes `07_monitoring_and_incidents.md` and `06_technical_documentation/` regenerate rather than rot, and it is worth pointing the same wiring at every other file in this tree, not only the two Module 9 built it for.

AS OF MID-2026

Three things in the surrounding legal picture are still moving as this course is being written, and they are worth a standing watch rather than a one-time check. The 2026 Digital Omnibus reached full political agreement in the first half of 2026 but had not yet been published in the Official Journal as of early July; the regulation number and the exact in-force dates it fixes are only settled once that publication happens, and Module 1 already flagged this as a "check before you cite it" item. The Article 6(5) classification guidelines are still in draft, published 19 May 2026 with consultation running to 23 July 2026 and the final version expected by the end of 2026; Module 2's classification memo should get revisited against the final text once it lands, especially for any ambiguous case reasoned from the draft's worked examples. And the Article 72(3) post-market monitoring plan template the Commission was due to adopt by 2 February 2026 had not appeared as adopted guidance by the time Module 9 was written; if that changes, the monitoring plan section of this pack is the one to update first.

Beyond those three, the Commission's AI Act Service Desk is the standing place to watch for new official templates and guidance as this area matures; a compliance practice this young will keep getting sharper edges.

None of that is a reason to wait before starting. Compliance, on the evidence this course has spent nine modules building, keeps being true only as long as the system producing it keeps running; treating it as a project with a finish line is where packs start decaying. You already have the machine that keeps it true: the logs from Module 6, the evals from Module 4 and Module 5, the oversight measurement from Module 7, and documentation that regenerates from source instead of being edited by hand under deadline pressure. Module 9 closed its own chapter by calling the shift from artifact to running system "the arc, closed." This module is where you point that same arc at a system of your own, and keep it closed.