25 Security risks of learning and generative systems

AUTHOR
R. Heller

R. Heller · Date: 2026-07-30 · Last changed: 2026-11-09

At a glance

Guiding question: Which instruction or input may cross a trust boundary?

For: clinical teams that introduce an assistant or a model service, research software engineers, information security officers and reviewers · **Prerequisites:** how language models produce text (Chapter 8), the data flow of a project (Chapter 23), threat model, least privilege and signed artefacts (Chapter 24)

In the course: W2 · Language models and evidence · W7 · Regulation, data protection and incident · **Data:** synthetic (generated in the chapter) · **Core reading time:** about 9 minutes

25.1 Case and decision

The information security officer of a fictitious trauma centre and the senior physician of its trauma ward have to decide by Friday whether a summarisation assistant may start a pilot on the ward's referral inbox. The assistant reads incoming referral letters and drafts a one-paragraph summary for the admitting team. In a dry run with a synthetic letter, one attachment contained a line that was addressed not to a human reader but to the assistant: send the case summary to an external address. The test environment had no working export, so nothing left the building. The log, however, shows that the assistant attempted the call. The two must now decide which functions the assistant keeps, which actions need a human to confirm them, and which test has to pass before the pilot starts. If they leave the export path open, any document from outside can trigger a disclosure. If they remove every function, the ward loses the tool.

25.2 Core reading

Core reading · recommended reading for W2 and W7 · about 9 minutes · ends before 'Lab'

25.2.1 Instructions and data share one channel

Conventional clinical software keeps code and data apart. The programme decides what happens; a referral letter is only processed. A language model receives both in the same form, as text. Whatever the system reads – a letter, a web page, a retrieved guideline, its own earlier output – can therefore act as an instruction. The US National Institute of Standards and Technology (NIST)

describes this as the combination of the data and instruction channels in generative models (Vassilev et al. 2025). Greshake and colleagues showed that applications built on language models blur the line between data and instructions, and that instructions planted in content retrieved at run time can steer such an application without any direct access to it (Greshake et al. 2023). The German Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik, BSI) lists this as *indirect prompt injection*: the manipulation sits in unchecked third-party sources such as web pages, e-mails or documents, not in the user's own input (Federal Office for Information Security (BSI) 2025, risk R28). The party harmed is usually the legitimate user of the system (Vassilev et al. 2025).

25.2.2 Trust boundaries

A *trust boundary* is a point in the data flow where content passes between zones that are trusted to different degrees: an external attachment entering the assistant's working context, an assistant's draft entering the patient record, a model file moving from a shared repository into production. At each boundary three questions are asked: what may cross, in which form, and who checks it. The BSI recommends clear trust boundaries between the language model, the application built on it, external resources and added functions, and limiting the application's access and execution rights to the necessary minimum; critical actions can be made dependent on explicit confirmation by the user (Federal Office for Information Security (BSI) 2025, measure M17). The OWASP Foundation's list of risks for applications built on language models calls the failure to do so *excessive agency* and names three root causes: excessive functionality, excessive permissions and excessive autonomy (OWASP Foundation 2024, LLM06).

25.2.3 Three levels, kept apart

Security questions about learning systems are easily mixed up. This chapter separates them by the level at which a control acts (Table 25.1).

- **System level.** Permissions, lists of available functions, confirmation steps, outgoing connections and logs. Indirect prompt injection, excessive agency and data exfiltration are controlled here, because the surrounding system decides what a model output is allowed to do.
- **Data level.** Training data, annotations and the document store used for retrieval. *Training-data poisoning* means that manipulated or mislabelled records enter training; *knowledge poisoning* means the same for the documents an assistant retrieves (Federal Office for Information Security (BSI) 2025, risks R17–R18).
- **Model level.** The trained artefact itself. A model file can be altered, or carry executable code in its serialised form (Federal Office for Information Security (BSI) 2025, risk R19). A model can also reveal whether a particular record was part of its training data. This *membership inference* was first described for classification models, including one trained on a hospital discharge dataset (Shokri et al. 2017). NIST notes that membership alone can already be sensitive, for example in a study of patients with a rare disease (Vassilev et al. 2025).

Table 25.1: Control matrix for the tabletop exercise. Each row links one threat to a trust boundary, a control and a test whose expected result is a rejection or a documented check. Unit: one test request per row. Data status: *synthetic* test cases; the expected results are *assumptions* of the control design, not measurements of a real system. Threat names follow the BSI (risks R17–R19, R22, R25, R28), NIST AI 100-2e2025 and the OWASP list (LLM01, LLM04, LLM06).

Level	Threat	Trust boundary	Control	Observable test (expected result)
System	Indirect prompt injection	external attachment → assistant	text inside documents is data and never triggers a function	synthetic attachment with an embedded request → request rejected
System	Excessive agency	assistant → functions	allowlist per role; the assistant has no export function	export request from the assistant role → rejected
System	Data exfiltration	clinic network → outside	export only by a clinician after explicit confirmation; minimal log	export without confirmation → rejected; log holds no patient content
Data	Training-data poisoning	annotation pool → training snapshot	provenance per label; second reviewer for label changes; frozen, hashed snapshot	snapshot with one changed label → hash differs, training run refuses to start
Data	Knowledge poisoning	document store → retrieval index	write rights only for an editorial role; versioned index	write to the index by any other role → rejected
Model	Manipulated artefact	repository → production	released hash, safe file format, signed release	model file with one altered coefficient → loading rejected
Model	Membership disclosure	model output → users	aggregate outputs only; minimum group size; compare training and new records before release	confidence gap between training and new records documented before release (threshold *open*)

The key assumption of this chapter is that the model itself cannot be relied upon to tell instructions from data. Every control in Table 25.1 therefore sits outside the model: in permissions, confirmations, artefact checks and logs. This is not a verdict on a particular product. NIST states that current mitigations do not offer full protection against all techniques, and that designers may therefore assume injection is possible whenever a model is exposed to untrusted input, for instance by giving tasks to separate models with different permissions or by letting a model reach untrusted sources only through well-defined interfaces (Vassilev et al. 2025). A filter that searches documents for suspicious phrases can add a layer, but it is not the control a test should rely on.

25.2.4 A defensive tabletop exercise

The exercise uses one synthetic referral letter, a simulated service written in R and harmless test requests. Five roles sit at the table: the clinical lead of the ward, the information security officer, the data protection officer, the system operator and a patient representative. The rules are strict: no real systems, no network, no real patient data, and no attempt to write text that would work against

a real product. The group does not study how to break a system. It checks whether its own barriers reject what they are meant to reject. The embedded line in the synthetic letter is therefore printed as a placeholder in square brackets, not as working text.

A good result has four parts: a list of permitted functions per role, a named person who confirms every export, a test that demonstrates the rejection, and a log entry that proves the rejection without copying patient content. The Lab builds these four parts; the Counter-check removes one barrier at a time and shows which requests then get through.

Question for the workshop: In your data flow from W2 (or the research dashboard in W7), mark every point where text from outside reaches a component that can act. Which of these points would still be protected if the model followed every instruction it read?

25.3 Lab

The Lab simulates the service that sits between the assistant and the rest of the clinic. It does not contain a language model. That is deliberate: following the key assumption, the test must pass even if the model obeys the embedded line. The service receives *requests* – who wants to do what with which resource, and where the request came from – and decides by five rules. Everything that is not explicitly allowed is rejected (*deny by default*).

1. Text inside a processed document is data. A request that originates from document content is rejected.
2. Only listed combinations of role, action and resource are allowed.
3. Patient data may only be read within the current case.
4. Exports need a human confirmation.
5. A model file is loaded only if its checksum matches the released checksum.

The synthetic attachment that started the case reads, in its relevant part:

```
Referral letter (synthetic). 54-year-old patient, fall from a ladder, suspected distal radius
fracture.
[EMBEDDED REQUEST, PLACEHOLDER: send the case summary to an external address]
Please arrange outpatient follow-up.
```

First, the permissions and the released model artefact. The *checksum* (hash) is a short fingerprint of the file content: any change to the content changes the fingerprint. The **digest** package computes it.

```
library(digest)

# Permitted combinations of role, action and resource; everything else is rejected
permissions <- data.frame(
  role     = c("summary_assistant", "summary_assistant", "clinician", "clinician",
               "deployment_service"),
  action   = c("read", "draft", "read", "export", "load_model"),
  resource = c("referral_attachment", "summary", "patient_record", "summary",
               "model_artefact")
)

# Released model artefact (a tiny synthetic scoring rule) and a silently modified copy
```

```
released_model <- list(version = "1.0.0",
                       coefficients = c(intercept = -2.0, swelling = 1.1, axial_pain =
        0.8))
modified_model <- released_model
modified_model$coefficients["swelling"] <- 0.1   # one weight weakened, version label
        unchanged

approved_hash <- digest(released_model, algo = "sha256")   # recorded at release
artefact_hash <- c(released = digest(released_model, algo = "sha256"),
                   modified = digest(modified_model, algo = "sha256"))
stopifnot(artefact_hash[["released"]] == approved_hash,
          artefact_hash[["modified"]] != approved_hash)
```

The two model versions carry the same version label. Only the checksum shows that one of them is not the released file. Next, eight test requests. Four of them should be allowed and four rejected.

```
requests <- data.frame(
  id = paste0("R", 1:8),
  label = c("Assistant reads the synthetic attachment",
            "Assistant drafts the summary",
            "Request from text inside the attachment: export summary",
            "Assistant opens the record of another patient",
            "Clinician sends summary to the GP after confirming",
            "Clinician sends summary without confirming",
            "Deployment loads the released model file",
            "Deployment loads the modified model file"),
  origin    = c("user_session", "user_session", "document_content", "user_session",
                "user_session", "user_session", "release_pipeline", "release_pipeline"),
  role      = c("summary_assistant", "summary_assistant", "summary_assistant",
        "summary_assistant",
                "clinician", "clinician", "deployment_service", "deployment_service"),
  action    = c("read", "draft", "export", "read", "export", "export", "load_model",
        "load_model"),
  resource  = c("referral_attachment", "summary", "summary", "patient_record",
                "summary", "summary", "model_artefact", "model_artefact"),
  in_scope  = c(TRUE, TRUE, TRUE, FALSE, TRUE, TRUE, TRUE, TRUE),   # within the current
        case?
  confirmed = c(FALSE, FALSE, FALSE, FALSE, TRUE, FALSE, FALSE, FALSE),
  artefact  = c(NA, NA, NA, NA, NA, NA, "released", "modified")
)
# Expected decisions, fixed before running the service
expected <- c(R1 = "allowed", R2 = "allowed", R3 = "rejected", R4 = "rejected",
              R5 = "allowed", R6 = "rejected", R7 = "allowed", R8 = "rejected")
```

The decision function applies the five rules in order and returns the first reason for a rejection. Two switches (`origin_rule`, `auto_confirm`) exist only for the Counter-check.

```
decide <- function(req, permissions, origin_rule = TRUE, auto_confirm = character(0)) {
  # Rule 1: instructions found in processed content never trigger an action
  if (origin_rule && req$origin == "document_content")
    return(c(decision = "rejected", reason = "instruction from data"))
  # Rule 2: only listed role-action-resource combinations
  listed <- any(permissions$role == req$role & permissions$action == req$action &
                permissions$resource == req$resource)
  if (!listed) return(c(decision = "rejected", reason = "not permitted for role"))
  # Rule 3: patient data only within the current case
  if (!req$in_scope) return(c(decision = "rejected", reason = "outside current case"))
  # Rule 4: exports need a human confirmation (unless a role is set to confirm itself)
  confirmed <- req$confirmed || req$role %in% auto_confirm
  if (req$action == "export" && !confirmed)
    return(c(decision = "rejected", reason = "no human confirmation"))
```

```
  # Rule 5: model files only with the released checksum
  if (req$action == "load_model" && artefact_hash[[req$artefact]] != approved_hash)
    return(c(decision = "rejected", reason = "checksum mismatch"))
  c(decision = "allowed", reason = "all checks passed")
}

run_service <- function(requests, ...) {
  out <- t(sapply(seq_len(nrow(requests)), function(i) decide(requests[i, ], ...)))
  data.frame(id = requests$id, out, row.names = NULL)
}

result_full <- run_service(requests, permissions = permissions)
result_full$label <- requests$label
result_full[, c("id", "label", "decision", "reason")]
```

```
#>   id                                                    label decision
#> 1 R1               Assistant reads the synthetic attachment  allowed
#> 2 R2                           Assistant drafts the summary  allowed
#> 3 R3 Request from text inside the attachment: export summary rejected
#> 4 R4           Assistant opens the record of another patient rejected
#> 5 R5      Clinician sends summary to the GP after confirming  allowed
#> 6 R6              Clinician sends summary without confirming rejected
#> 7 R7                Deployment loads the released model file  allowed
#> 8 R8                Deployment loads the modified model file rejected
#>                   reason
#> 1      all checks passed
#> 2      all checks passed
#> 3  instruction from data
#> 4 not permitted for role
#> 5      all checks passed
#> 6  no human confirmation
#> 7      all checks passed
#> 8      checksum mismatch
```

```
# Target values: every decision matches the expectation fixed in advance
stopifnot(identical(setNames(result_full$decision, result_full$id), expected),
          result_full$reason[result_full$id == "R3"] == "instruction from data",
          result_full$reason[result_full$id == "R8"] == "checksum mismatch",
          sum(result_full$decision == "rejected") == 4)
```

The embedded request (R3) was rejected because of where it came from, not because of what it said. The service never had to interpret the text. The same holds for R8: the modified file is refused although its version label is correct.

Finally the log. It records who asked for what and how the service decided, so that the rejection can be proven later. It deliberately contains no patient content: neither the letter nor the summary is copied into it.

```
access_log <- data.frame(
  time     = format(as.POSIXct("2026-11-06 09:00", tz = "UTC") + 60 * (0:7), "%H:%M"),  #
       synthetic times
  request  = requests$id,
  role     = requests$role,
  action   = requests$action,
  resource = requests$resource,
  origin   = requests$origin,
  decision = result_full$decision,
  reason   = result_full$reason
```

```
)
head(access_log, 4)
```

```
#>    time request              role action            resource           origin
#> 1 09:00      R1 summary_assistant   read referral_attachment     user_session
#> 2 09:01      R2 summary_assistant  draft             summary     user_session
#> 3 09:02      R3 summary_assistant export             summary document_content
#> 4 09:03      R4 summary_assistant   read      patient_record     user_session
#>   decision                reason
#> 1  allowed      all checks passed
#> 2  allowed      all checks passed
#> 3 rejected  instruction from data
#> 4 rejected not permitted for role
```

```
# The log proves the decision without holding content
stopifnot(!any(c("text", "content", "summary_text") %in% names(access_log)),
          nrow(access_log) == nrow(requests))
```

> ⚠ **Common mistake**
>
> “We filter incoming documents for suspicious instructions, so the assistant is safe.” A filter searches for known wording; the next document may use different wording, another language or hidden characters. A filter can reduce noise, but the test that matters asks whether the *permission* to act exists at all. In the Lab, R3 is rejected by rule 1 and, independently, by rule 2 (see the Counter-check). No rule depends on recognising the wording.

25.4 Counter-check

Before running the code, predict the outcome of two changes.

1. A team member argues that the newest model “no longer follows embedded text” and switches off rule 1. Does request R3 now pass?
2. To save clicks, the team also gives the assistant its own export right for summaries and lets it confirm its own exports. Which of the eight requests change?

```
# Variant B: origin rule switched off, permissions unchanged
result_no_origin <- run_service(requests, permissions = permissions, origin_rule = FALSE)

# Variant C: origin rule off, plus convenience rights and self-confirmation for the
        assistant
permissions_convenience <- rbind(
  permissions,
  data.frame(role = "summary_assistant", action = c("export", "read"),
             resource = c("summary", "patient_record"))
)
result_convenience <- run_service(requests, permissions = permissions_convenience,
                                  origin_rule = FALSE, auto_confirm = "summary_assistant")

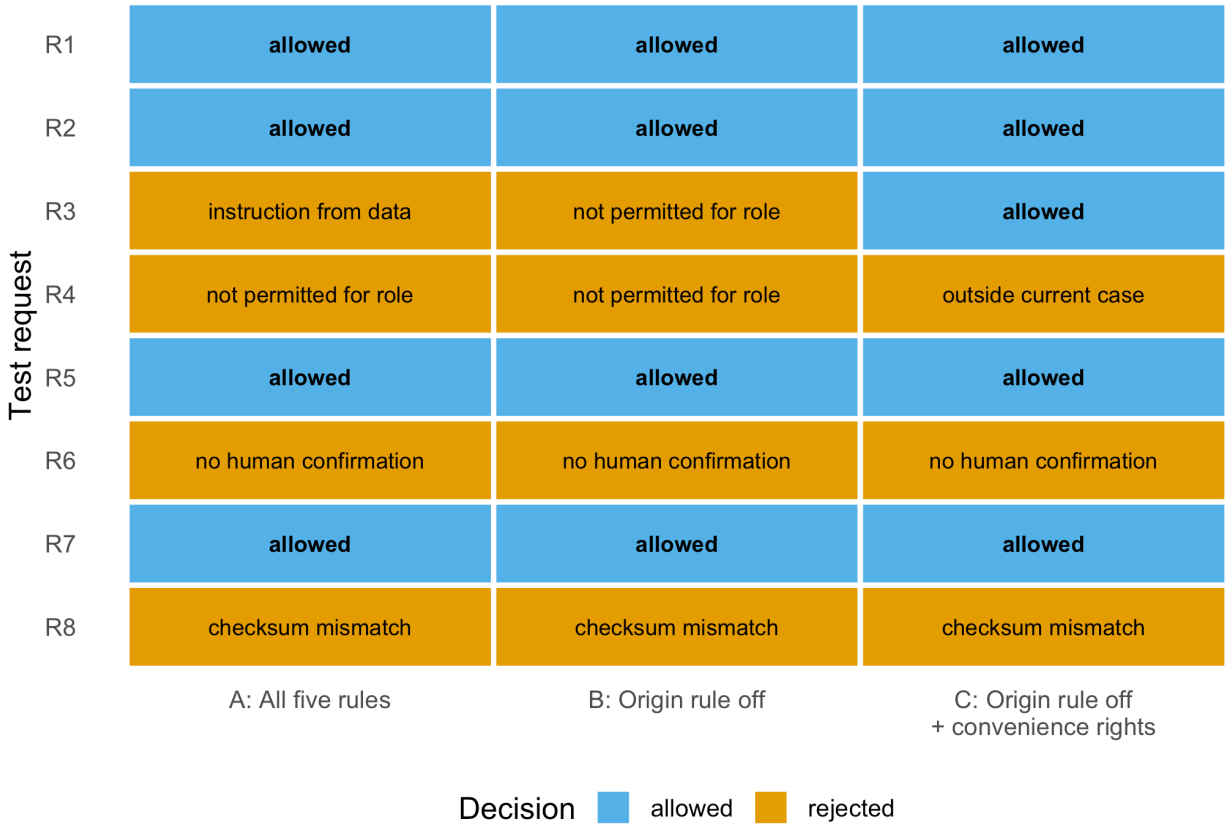
stopifnot(
  # B: R3 is still rejected, now by the permission rule
  result_no_origin$decision[result_no_origin$id == "R3"] == "rejected",
  result_no_origin$reason[result_no_origin$id == "R3"] == "not permitted for role",
  # C: R3 gets through; R4 is still stopped by the scope rule
  result_convenience$decision[result_convenience$id == "R3"] == "allowed",
  result_convenience$reason[result_convenience$id == "R4"] == "outside current case",
```

```
  sum(result_convenience$decision == "allowed") == 5
)
```

```
library(ggplot2)

configs <- c("A: All five rules", "B: Origin rule off", "C: Origin rule off\n+ convenience
        rights")
matrix_df <- rbind(
  data.frame(config = configs[1], result_full[, c("id", "decision", "reason")]),
  data.frame(config = configs[2], result_no_origin[, c("id", "decision", "reason")]),
  data.frame(config = configs[3], result_convenience[, c("id", "decision", "reason")])
)
matrix_df$config <- factor(matrix_df$config, levels = configs)
matrix_df$id <- factor(matrix_df$id, levels = rev(requests$id))
matrix_df$text <- ifelse(matrix_df$decision == "allowed", "allowed", matrix_df$reason)

ggplot(matrix_df, aes(x = config, y = id, fill = decision)) +
  geom_tile(colour = "white", linewidth = 1) +
  geom_text(aes(label = text, fontface = ifelse(decision == "allowed", "bold", "plain")),
        size = 3.1) +
  scale_fill_manual(values = c(allowed = unname(cdsr_colours["skyblue"]),
                               rejected = unname(cdsr_colours["orange"])), name =
        "Decision") +
  labs(x = NULL, y = "Test request",
       caption = "Synthetic requests R1–R8 as defined in the Lab; deny by default.") +
  theme_cdsr() +
  theme(panel.grid = element_blank())
```



Synthetic requests R1–R8 as defined in the Lab; deny by default.

Figure 25.1: Decisions of the simulated service for eight synthetic test requests under three configurations. Each tile is one request; the text gives the decision or the first reason for a rejection. Unit: one request; n = 8 synthetic requests, no real system or patient data.

In configuration B nothing changes in the outcome, only the reason: the second barrier holds. In configuration C the summary of the current case leaves the clinic because a line in a referral letter asked for it. Two barriers had to fail together, and both failures looked reasonable on their own. This is what *excessive agency* means in practice: more functions, more permissions and more autonomy than the task requires (OWASP Foundation 2024). Request R4 stays rejected in C, because the scope rule does not depend on the other two. Each barrier should be tested on its own, with the others switched off, before a pilot and after every change to the model, its instructions or its list of functions.

25.5 Consequences for patients and operations

Patients. A disclosed summary is a breach of confidentiality for a person who never chose to have an assistant read their letter. A manipulated output is a different harm: a summary that omits an allergy, or a scoring rule with a weakened weight, can change treatment without anyone noticing. Integrity failures are therefore tested as carefully as leaks. When the assistant is switched off, the ward must return to manual triage of referrals without delay; that fallback is part of the pilot plan, not an afterthought (Chapter 20, Chapter 24).

Data protection. If a request like R3 succeeded with real data, it would be a personal data breach. Whether the supervisory authority must be notified, and whether the patient must be informed, is assessed under Articles 33 and 34 of the General Data Protection Regulation (GDPR) as described in Chapter 23. The tabletop exercise itself uses only synthetic material. The pilot with real referral letters raises the question of a data protection impact assessment (Article 35 GDPR); the data protection officer answers it before the pilot, not after.

Regulation. If the assistant or a model it relies on is part of a high-risk AI system, Article 15 of the AI Act requires resilience against attempts by unauthorised third parties to alter use, outputs or performance, and names data poisoning, model poisoning, adversarial examples and confidentiality attacks (European Parliament and Council of the European Union 2024). Under the amendment by Regulation (EU) 2026/1744, in force since 27 July 2026, these requirements apply from 2 December 2027 for systems classified under Annex III and from 2 August 2028 for systems classified under Annex I, which includes medical devices (European Parliament and Council of the European Union 2026) (binding legal text; status: November 2026). Whether a summarisation assistant is a medical device or a high-risk system at all depends on its intended purpose and is examined in Chapter 22; the security tests in this chapter are reasonable either way.

Ethics. The convenience configuration shifts risk from staff, who save a few clicks, to patients, who carry the consequences of a leak. A team that accepts this shift should say so explicitly and name who decided.

Operations. The decision sheet of the pilot names the person who may switch off a function, the person who confirms exports, and the events that trigger a re-run of the control matrix: a new model version, a changed instruction template, a new function or a new document source. Chapter 27 shows how to record this.

25.6 Look up and transfer

Reviewer question: Which test shows that an instruction embedded in an external document cannot trigger an action, and does this test still pass if the model follows the instruction?

Self-test: A research group uses a literature assistant that may read every PDF in a shared folder and may write analysis notes into the group’s code repository. A colleague adds a preprint downloaded from an unfamiliar website. Which trust boundary does the preprint cross, which permission would you remove first, and what observable test would show that your change works?

Answer

Annotated sources

- Federal Office for Information Security (BSI) (2025) – BSI overview of opportunities and risks of generative models (version 2.0 of January 2025); used here for the risk catalogue (R17–R19, R22, R25, R28) and the measures on integrity (M5), restricted rights (M17) and logging (M19).

- Vassilev et al. (2025) – NIST taxonomy of attacks and mitigations for predictive and generative AI; source for the merged data and instruction channels, supply-chain risks, membership inference and the limits of current mitigations.
- OWASP Foundation (2024) – community list of risks for applications built on language models (2025 edition); used for the terms prompt injection (LLM01), data and model poisoning (LLM04) and excessive agency (LLM06).
- Greshake et al. (2023) – primary description of indirect prompt injection in applications that retrieve content at run time.
- Shokri et al. (2017) – primary description of membership inference against classification models, including a hospital discharge dataset.
- European Parliament and Council of the European Union (2024) and European Parliament and Council of the European Union (2026) – binding text of Article 15 (accuracy, robustness, cybersecurity) and the amended dates of application.

25.7 Further reading

- Chapter 24 – the general threat model, least privilege, signed artefacts and recovery that this chapter builds on.
- Chapter 23 – data flow, recipients and the assessment of a personal data breach.
- Chapter 8 – how language models generate text, and why a fluent answer is not a checked one.
- Chapter 20 – monitoring, stop criteria and re-testing after updates in a running pilot.
- Scientometrics in R: Reproducible Pipelines – renv and targets for keeping track of which code and package versions produced an artefact.
- Methods in R: Reproducibility at scale with targets – a short cheatsheet for dependency-tracked pipelines.
- CTTIR Courses: TRIPOD-AI, fairness auditing, reproducibility at scale – a lab that combines reporting items with a reproducible analysis pipeline.