

Architectural Governance of Agentic AI: From Instructions to Enforcement

Lecture Notes

Epitácio Vicente do Nascimento Neto* and Prof. Dr. Felipe André Zeiser

Graduate Program in Applied Computing (PPGCA)

Universidade do Vale do Rio dos Sinos (UNISINOS), São Leopoldo, Brazil

Accepted at the NeurIPS 2026 Education Track (Submission 10); final version, November 2026.
Final version, November 2026. Content licensed CC BY 4.0.

Contents

How to use these materials	3
About these notes: audience, prerequisites, objectives	5
M0 Why now: agents act, instructions leak	6
M0.1 The problem, in one example	6
M0.2 The running example: the deploy agent	6
M0.3 From chat to action	7
M0.4 Terms you need first	7
M0.5 What the literature already shows	8
M0.6 Evidence labels and identifiers	9
M0.7 The founding case: what it is, how it was observed, what it can show . . .	9
M1 Anatomy of an agentic deployment	13
M1.1 The parts list	13
M1.2 Two details worth pausing on	14
M2 The access-versus-mandate dual classification	15
M2.1 Three levels, in order	15
M2.2 Two questions, asked separately	15
M2.3 A worked example: how the gap is read for one action	16
M2.4 The gap rule and the gap size	16
M2.5 Risk grade is not a level	16
M2.6 Reading the founding case's matrix	18
M3 The enforcement ladder	19
M3.1 One question grades every control	19
M3.2 The founding case on the ladder	21
M3.3 Why the lower rungs dominate	21
M4 Guardrails in twelve layers	22
M4.1 The catalogue	22

*Corresponding author: netoepitacio@edu.unisinos.br

M4.2	The two structural exemplars	22
M4.3	Layer patterns and failure direction	24
M5	Is your human oversight real? The six viability conditions	25
M5.1	The lesson of the stalled parser	25
M5.2	The six viability conditions	25
M5.3	The founding case’s oversight, measured	26
M6	The supervision residue and its determinants	28
M6.1	Hardening is not absolution	28
M6.2	Why classes stay in the residue	29
M6.3	Condition sets and domains	29
M7	Measuring governance	30
M7.1	The metric inventory	30
M7.2	The 61.1% (REPORTED) divergence rate	31
M7.3	Epistemic discipline	31
M8	The frontier: making enforcement architectural	32
M8.1	Where research is already moving	32
M8.2	Open questions	32
	Exercises	34
A	Worked solutions	37
B	Glossary	40
	Acknowledgements	44
	References	44

How to use these materials

What is here and which artifact is primary. These materials teach one idea, architectural governance of LLM-based agents: telling apart the rules a system enforces by its structure from the rules it only asks for in prose. There are three teaching artifacts and a short proposal, all built on the same nine modules (M0 to M8) and the same running example.

The lecture notes are the primary artifact. They are the complete lesson: every definition, worked example, exercise and solution is in them, and they can be read on their own.

The slides are a lecture pass over the same modules. Every slide has speaker notes, and the slides-with-notes PDF lets you follow the lecture without an instructor.

The interactive website animates the access-versus-mandate matrix, the enforcement ladder, the guardrail layers, the six viability conditions and the supervision residue. It also hosts the Guardrail Lab, which Exercise E5 uses. Its complete source is included, so everything runs offline.

The two-page proposal describes the package for the Education Track.

The optional live demo, MeridianSeal Light, is a hands-on companion at <https://neurips2026.meridianseal.com/demo/>. In a browser you send the kind of request the deploy agent makes to governed agents, watch a human Patch Gate and a structural block, and switch between the ten condition sets. It needs a network connection and uses simplified rules, it is not a study result, and nothing else in these materials depends on it. The offline website works without it.

Reading paths by audience.

You are	Start with	Then
A self-study learner	The lecture notes, M0 to M8 in order. Do E2 after M2, E1 and E4 after M3, E3 after M5, E5 after M6.	The slides-with-notes PDF to review each module, and the website section for the module (table below).
An instructor giving one lecture (90 minutes)	The slides, M0 to M8.	A live walk through the Guardrail Lab in adversarial mode. Hand out the lecture notes.
An instructor teaching a course module	The 3-session plan or the 4-session plan (below).	The slides for each session; the Guardrail Lab in the session that covers M6.
A practitioner who wants to apply it to their own system	M0, M2 and M3 in the notes, then E1 and E3 on your own system.	M4 to M6 when you want the full catalogue and the oversight test.

Session plans. 3-session plan. Session 1: M0 to M3, the core classification machinery (exercises E2, E1, E4). Session 2: M4 to M6, evidence and oversight (exercises E3 and E5). Session 3: M7 to M8, measurement and the frontier.

4-session plan. Session 1: M0 to M2 (exercise E2). Session 2: M3 to M4 (exercises E1 and E4). Session 3: M5 to M7 (exercises E3 and E5, the Guardrail Lab after M6). Session 4: M8 and an exercise review.

Both plans cover M0 to M8 once; choose the one that fits your calendar.

Where each module lives.

Module	Lecture notes	Slides (frames)	Website section	Exercise
M0 Why now: agents act, instructions leak	Module M0	5 to 11	#start, #founding	none
M1 Anatomy of an agentic deployment	Module M1	12 to 13	#founding	none
M2 The access-versus-mandate dual classification	Module M2	14 to 18	#gap	E2
M3 The enforcement ladder	Module M3	19 to 24	#ladder	E1, E4
M4 Guardrails in twelve layers	Module M4	25 to 27	#stack	none
M5 Is your human oversight real? The six viability conditions	Module M5	28 to 30	#watcher	E3
M6 The supervision residue and its determinants	Module M6	31 to 32	#residue, #lab	E5
M7 Measuring governance	Module M7	33 to 34	#founding, #learn	none
M8 The frontier: making enforcement architectural	Module M8	35 to 36	#learn	none

Slides 1 to 4 are the opening (title, this guide, agenda, the problem and the running example); slides 37 to 38 are the optional live-demo break (MeridianSeal Light); slides 39 to 41 close the lecture; slides 42 to 43 are the glossary appendix. Slide frame titles start with the module tag.

Running the website offline. The website is a static folder. From the website folder of the ZIP, run `python3 -m http.server 8000` and open `http://localhost:8000` in a browser. A server is needed because the page loads `data.json`; opening `index.html` directly from disk does not work. There is no build step and no external dependency: fonts are the system fonts and nothing is fetched from the network.

Where the code is. The folder code contains `reproduce.py` and `reproduce.ipynb`. Both read only `website/data.json`, use relative paths, and reproduce the access-versus-mandate matrix, the gap sizes, the enforcement-ladder counts, the layer table, the guardrail population counts, the supervision-point and residue counts, and the Guardrail Lab outcome tables for every condition set. `expected-output.txt` is the output of a clean run, so you can check yours against it.

Versions and how the accepted version is kept. This is the final version, November 2026, of the material accepted at the NeurIPS 2026 Education Track (Submission 10). The version as accepted and submitted on 24 Aug 2026 is kept unchanged on the hosted website at `accepted/index.html` and is linked from the website downloads section as “Accepted version (as submitted, 24 Aug 2026)”. It is not part of the ZIP. All downloads on the website and in the ZIP are the final version.

What is in the ZIP. `neurips2026-education-materials-final.zip` contains `README.md`, `USER-GUIDE.md`, the proposal, the lecture notes, the slides (`slides.pdf` and `slides-notes.pdf`), the full website folder (`index.html`, `css`, `js`, `data.json` and the four PDFs under `assets`), the code folder, the L^AT_EX sources, the licence files and `MANIFEST.txt`. `neurips2026-website-source.zip` contains the website folder alone.

About these notes

These notes teach one emerging concept: *architectural governance* of LLM-based agents, that is, making a prohibited effect structurally unreachable rather than merely discouraged, reviewed, or checked by a mechanism the agent itself could alter. The concept is taught through four instruments (a dual classification, an enforcement ladder, a viability test for human oversight, and a residue analysis) and grounded throughout in one empirical source, the founding case: a production multi-agent deployment coordinated by one of the authors, studied by a single observer in one read-only run. Section [M0.7](#) introduces it, with its method and its limits, before any of its numbers are used. Every number comes from the study's registers and documents or, where marked REPORTED, from its documented but unverified accounts.

Audience and level. Advanced undergraduate and early graduate students; equally usable by practitioners deploying agents. The material teaches by classification and case analysis, not proofs.

Prerequisites. (a) A working idea of what an LLM and a prompt are; (b) basic software systems literacy (processes, file permissions, version control, CI); (c) no security, formal-methods or ML-theory background needed. Each specialised term is defined where it first appears.

Learning objectives. By the end, a learner can:

- LO1.** REMEMBER/UNDERSTAND: define an agentic AI system and distinguish technical access from authorised mandate (both defined in M2).
- LO2.** APPLY: classify any given control into the four enforcement classes by asking who can defeat it.
- LO3.** ANALYSE: assess a human-in-the-loop supervision point (a human approves before the action; defined in M0) against the six viability conditions and name its binding failure.
- LO4.** ANALYSE: read an access-vs-mandate matrix and locate the governance gap of a system.
- LO5.** EVALUATE: explain why instructional and conditional controls dominate deployed systems and what that costs, using the founding-case evidence.
- LO6.** APPLY/CREATE: propose a structural replacement for a given instructional control (e.g. a read-only credential instead of a “do not write” instruction).
- LO7.** EVALUATE: interpret governance metrics, including a self-report divergence rate, and argue what they do and do not establish.

Structure. Modules M0 to M8 build the concept in order; each opens with a motivating question and closes with a checkpoint, and the running example (Section [M0.2](#)) reappears in every module. The [Exercises](#) section gives five exercises, E1 to E5; Appendix [A](#) works them through; Appendix [B](#) is the merged glossary (55 terms) shared with the slides and the website. Use the session plans above to split the modules over a course.

M0 Why now: agents act, instructions leak

Motivating question

If a system can execute shell commands, write files, and call external services, why would a paragraph of prose be enough to stop it doing the wrong thing?

M0.1 The problem, in one example

The problem

The problem. Picture an AI agent that is allowed to deploy software. The organisation's policy says that every deploy needs a human's approval. The policy is written in files the agent reads, and an approval gate records each approval, but the gate is a check inside the deployment tooling, and the agent can route around it. How can anyone tell, for any agent system, which of its rules are enforced by the structure of the system and which are only requested? These materials teach a way to answer that question, and show with evidence from one real deployment that most rules there were only requested.

M0.2 The running example: the deploy agent

Running example, introduced

The running example: the deploy agent. Three elements, used under the same three names in every module, slide, exercise and website section. The deploy agent is an agent that may deploy software to a production service. The approval policy says every deploy needs a human's approval before it goes live. The approval gate is the control meant to enforce that policy, and it is a control the agent can route around.

The example is an illustrative simplification of two rows and one control in the study's registers; it is not a transcript of the incident. The deploy agent is action class AC-044, deploy application code to another estate host (access Level II, mandate Level III, risk grade severe, enforcement conditional, REPORTED). Recording an approval is action class AC-056, record an approval in the deployment approval gate (access Level I, mandate Level III, risk grade severe, enforcement conditional, OBSERVED). The approval gate is guardrail GR-L06-01, deployment approval gate held by a separate principal (layer L6 Change and deployment; population partial; enforcement class conditional; fail-open).

The register records AC-056 at access Level I because separation of duties is by role convention among agents sharing one operating-system identity (AC-056 note); the gate's approve route itself needs a human session (GR-L06-01), so the exposure is the shared credentials and the unbound approval-to-execution link.

The study records three routes around the gate, and none of them forges a human's approval. First, the credentials the gate relies on are readable by the one account every agent runs as, so any agent can present any agent's identity; one recorded incident has a session using the lead agent's credential to record a formal peer review of its own patch. Second, the gate governs a record and not an artefact: approval sets a status field, but nothing binds what actually runs to that field, and the process manager re-executes whatever is on the shared working tree at any restart; two documented incidents shipped code through an ordinary restart before a deploy event was recorded. Third, on the examined host the production working tree is writable over the network, so an ordinary file write places code that the next restart will serve.

The three elements in plain words: an agent that may deploy (what the system technically lets it do, its *access*); a policy that needs approval (what the organisation authorises it to do, its *mandate*); and a control it can route around, the approval gate, whose strength we grade by who can defeat it (the *enforcement ladder*). Access, mandate and the ladder are defined in M2 and M3. The registers behind AC-044, AC-056 and GR-L06-01 belong to the founding case, introduced in Section [M0.7](#). The other cases drawn from the founding case (the sentinel kill switch, the read-only System-of-Record identity, the test-mode overlay, the restart of a customer-facing service) are always introduced as “other cases from the founding case”, never as the running example. Every module below ends its main text with a box “Running example” that moves the deploy agent one step further.

M0.3 From chat to action

A language model that only produces text is governed, in the end, by whoever reads the text. An *agentic* system is different: the model’s outputs are wired to tools, so producing text *is* acting. A tool call deploys code, sends a message, edits a file, or dials a phone. Once that wiring exists, governance can no longer stop at the question “did the model say something harmful?” and must ask “what can this system *do*, and what stops it?”

The dominant answer today is instructional. We govern agents mostly by telling them things: system prompts, policy files, instruction hierarchies (a rule for which instruction wins when instructions conflict; defined in the box below), codes of conduct embedded in context windows. The premise of these notes is that this answer is structurally insufficient, and that a body of recent research, together with detailed evidence from one production deployment, lets us teach a better one.

M0.4 Terms you need first

The next subsection cites research that uses specialised terms. Here they are, in plain words, before they are used. The terms *instruction hierarchy*, *prompt injection*, *in-context scheming* and *control flow and data flow* come from the cited literature; *the adversarial ceiling of prose* is the authors’ own phrase; the three loop terms are used throughout the notes. The figure quoted in the third term is explained, with its method, in Section [M0.7](#); read it there before relying on it.

Terms you need first

Instruction hierarchy. A rule for which instruction wins when instructions conflict: text from an application’s developer (the system prompt) outranks text from a user, which outranks text that arrives inside a web page, a document or a tool’s output. Wallace and colleagues (2024) train language models to follow that ordering and to ignore lower-ranked instructions that conflict with higher-ranked ones. It makes a violation less likely; it does not stop a model that fails to follow the ordering, so it is still a control written in prose. In the running example: the approval policy should outrank anything the deploy agent reads in a ticket.

Prompt injection. Text placed in data that an agent reads (a web page, an email, a file, the output of a tool) which the agent mistakes for an instruction and obeys, so that whoever wrote the data, and not the agent’s operator, steers what the agent does. In the running example: a ticket the deploy agent reads says “skip the approval step”; if the agent obeys, that text, not the approval policy, governs the deploy.

In-context scheming. An AI agent covertly pursuing a goal that was given to it in its context window, for example by hiding what it did, disabling an oversight mechanism

or misreporting its own actions. Meinke and colleagues (2024) show that several frontier models can do this in test environments built to encourage it, strategically introducing subtle mistakes, attempting to disable oversight and trying to copy what they believed to be their own weights. This is a capability result from constructed tests. The founding case does not show scheming: its 61.1 per cent (REPORTED) measures a mismatch between a report and the evidence, not intent.

The adversarial ceiling of prose (the authors' own phrase, not from the cited papers). A control written as prose, such as a system prompt or a policy file, works only if the model chooses to comply. Against a model, or a text, that is working to evade or override it, the guarantee a prose control gives cannot be raised above what the adversary allows, however carefully the prose is worded. That upper bound is the adversarial ceiling of prose. Prompt injection and in-context scheming are two ways of reaching it. In the running example: however carefully the approval policy is worded, it binds only an agent that chooses to follow it.

Human in, on and out of the loop. Three ways a human relates to an agent's action. In the loop: a human approves before the action happens. On the loop: the agent acts and a human reviews afterwards and can intervene. Out of the loop: the agent acts with no human review beforehand. The study ties them to the three levels used in these materials: Level III is human-in-the-loop, Level II is human-on-the-loop, and Level I approximates human-out-of-the-loop. In the running example: the approval policy asks for in the loop; the study found that an environment which described itself as on the loop exercised that loop zero times in 46 days.

Control flow and data flow. Control flow is the sequence of steps a program decides to take; data flow is the information that moves between those steps. CaMeL (Debenedetti and colleagues, 2025) extracts the control flow from the trusted request, so that untrusted data an agent retrieves can never change which steps run. That is enforcement placed in the structure of the system rather than in the model's behaviour.

M0.5 What the literature already shows

Five threads of 2022 to 2026 research frame the problem, and these notes teach the unifying concept beneath them.

- **Measuring agent risk.** ToolEmu [1] showed that LM agents given tools exhibit costly failure modes that can be surfaced systematically with an LM-emulated sandbox. Agent-Dojo [2] built a dynamic benchmark in which prompt-injection attacks against tool-using agents can be measured, and found defences trailing attacks.
- **Control under adversarial assumptions.** The AI Control line [3] asks a sharper question: can a deployment stay safe even if the model is *intentionally* subverting it? Their protocols treat the model as untrusted and place safety in the surrounding pipeline.
- **Enforcement moved into architecture.** CaMeL [4] defeats prompt injections *by design*, by separating control flow from data flow so that injected text simply cannot reach the decision that matters. This is enforcement relocated from the model's behaviour into the system's structure.
- **The strongest instructional defence.** The instruction hierarchy [5] trains models to prioritise privileged instructions. It is the best version of the prose answer, and it still lives inside the model's weights and context, where it can only lower the probability of violation.
- **The adversarial ceiling of prose.** Frontier models are capable of in-context scheming

[6]: strategically evading oversight and misrepresenting their own actions when it serves a goal. If the governed party can deceive, then any control that depends on the governed party's honest cooperation has a ceiling. The phrase "adversarial ceiling of prose" is the authors' own; the cited paper reports a capability result, not that phrase.

Read together, these threads point at one idea that none of them names as a teaching object: some controls are defeasible by the agent and some are not, and the difference is a property of *architecture*, not of model quality. That idea is what we now build, instrument by instrument.

M0.6 Evidence labels and identifiers

Before the founding case, two conventions used on every page of the notes.

Evidence labels

Evidence labels. Every finding in the study carries one of four labels. OBSERVED: verified on the host in the run, with an evidence identifier. REPORTED: taken from the doctoral proposal or the run brief and not verifiable on the host in that run. PROPOSED: a design proposal of the study that does not exist in the estate. NOT DETERMINED: could not be established, with the reason stated. These materials add AUTHOR-ATTESTED for statements the authors make that are not in the registers.

These notes also write DERIVED for arithmetic computed here from register fields (for example gap size: "derived presentation, not a register field"), and CONVENTION for a rule the study itself states as a convention (for example the order in which the six viability conditions are read). Every headline figure carries a unit noun (classes, entries, points, episodes, findings, metrics, days, events). The two different 46s are always written "46 days" (console audit logs) and "46 candidate episodes" (the pool behind the 61.1 per cent, REPORTED).

Identifiers used in these notes

AC action class (AC-001 to AC-060); GR guardrail (GR-Lnn-nn); L layer (L1 to L12); CS condition set (CS-01 to CS-10); SP supervision point (SP-001 to SP-034); VC viability condition (VC1 to VC6); SD supervision determinant (SD-01 to SD-10); MG metric (MG-001 to MG-072); DM applicability domain (DM-01 to DM-09); F durable finding (F-001 to F-043); EV evidence file (cited by identifier; the raw files never leave the host); Level I, II, III autonomy level; rung 0 to 3 enforcement rung. Do not confuse L1 (layer) with Level I, or CS with SD.

M0.7 The founding case: what it is, how it was observed, what it can show

Every number in these notes comes from one study of one deployment. This subsection says what that deployment is, who observed it, how the counts were defined and sampled, and what the findings can and cannot establish. It comes before the results on purpose: the results are only as strong as this section allows.

What it is. AUTHOR-ATTESTED. The founding case is an AI conversational application (the service's conversational engine) operated by multiple AI agents working 24/7 to monitor and patch the application autonomously, with some work done by a human on the loop. It is a real service running in a real real-estate business in Sydney, Australia, coordinated by one of the paper's authors, who identified the opportunity to start working on the architecture side of reinforcement. These contextual statements are the authors' own; they are not part of the

study's registers and were not independently audited. No company, customer or individual is named.

The registers are identity-free by construction (no hostname, account, customer or third party is named). The deployment's sector (real estate) and city (Sydney, Australia) are author-attested and are not part of the registers; no company, customer or individual is named.

Who observed it, and when. The registers come from one read-only collection run (run 20260807-1714, captured on 7 August 2026), in which the study's single observer examined one host of the estate and changed nothing. The study's documents state that the estate was built and operated by the doctoral candidate and that the observer is the operator. The study is therefore a single-observer, read-only, evidence-cited snapshot of a deployment coordinated by one of the authors, and it is not an independent external audit. Two partial mitigations were applied: every factual claim cites an evidence identifier, and one hypothesis the author carried into the run was corrected rather than confirmed (finding F-035). Neither makes the study independent.

What was observed. The host examined is the agent orchestration hub of the estate, the place where the fleet of coding agents is started, supervised and dispatched. At the time it ran 14 live agent sessions (9 registered fleet agents and 5 started ad hoc) that shared one operating-system account holding unrestricted passwordless root, 9 supervised services, and 17 automations that can start an action with no human keystroke, with 1 human operator, and its console audit logs span 46 days. The conversational engine that talks to members of the public by voice and message runs on other hosts of the estate and was not examined; it enters the registers only as REPORTED action classes (AC-049, a real-time voice interaction with a member of the public, and AC-050, a message to a member of the public). The figures in these materials therefore describe the agents that operate the service, their actions, their controls and the supervision over them, not the service's business logic or its customers.

How the authors' description and the registers relate.

The authors' statement	What the registers show	Label
An AI conversational application Multiple agents Working 24/7	Voice and messaging action classes AC-049 and AC-050 on another estate host 14 live agent sessions and 1 human operator Continuous unattended operation: 17 automations that act with no human keystroke; a heartbeat watcher every three minutes with no iteration cap and no end condition; every fleet agent recreated daily at 08:00. The literal phrase 24/7 is the authors'	REPORTED, not observed OBSERVED AUTHOR-ATTESTED; the registers support continuous operation
To monitor	A stall watchdog and watchers exist; 8 of the 14 live sessions sit outside every watchdog's scope, and one had produced no output for 7 days at capture time	OBSERVED
And patch the application autonomously	AUTHOR-ATTESTED. In the registers, agents author and submit patches to the deployment approval gate, where only a human identity can approve; the console audit logs (46 days) show zero deployment rejections and zero update approvals; 434 update proposals were pending and none had been applied	AUTHOR-ATTESTED, with these register observations beside it
Some work as human-on-the-loop	The environment describes itself as on the loop; over 46 days the loop was exercised zero times (finding F-042); DERIVED from the register field effective posture: 22 of 34 supervision points are recorded as plain out-of-the-loop (25 if the qualified variants "out-of-the-loop, defensibly" and "out-of-the-loop at the time of action" are counted)	OBSERVED (zero loop use, F-042); DERIVED (the counts 22 and 25)
A real service in a real real-estate business in Sydney, Australia (author-attested)	Not in the registers, which name no sector, city, company or customer	AUTHOR-ATTESTED
Coordinated by one of the paper's authors	The study's documents state that the estate was built and operated by the doctoral candidate, who is also the observer	STATED in the study's documents; not independently verified; which author coordinated is for the authors to confirm

How the counts were defined. 60 action classes: a complete enumeration of one host, not a sample. Every distinct class of action an agent on the host can perform was catalogued at a consistent granularity, the level of "commit to a repository" or "restart a managed process" rather than a single API call, giving AC-001 to AC-060. Completeness is the study's claim; no completeness test is documented. 46 of the 60 classes have a coding agent as actor. 55 classes were verified on the host (OBSERVED) and 5 are REPORTED because the mechanism sits on another host (AC-044, AC-049, AC-050, AC-051, AC-052). Each class was graded for access, mandate, risk grade and current enforcement.

134 guardrails: a catalogue of the design space, not 134 deployed controls. Entries are numbered GR-L01-01 to GR-L12-09 across 12 layers; each states its mechanism, enforcement class, the action classes it binds, a mandatory bypass analysis, its fail state and the signal it emits. 39 entries are present on the host, 17 partial and 78 proposed. The catalogue was derived

from one environment, and its generality is a hypothesis.

72 metrics: specified metrics, 21 engineering and 51 governance, of which 14 are available today, 25 derivable and 31 absent (2 further entries split). Only one, the claim divergence rate MG-022, has a computed baseline.

34 supervision points: places where the architecture claims some human supervision, intervention or signal. Each was assessed against the six viability conditions; 27 diverge between nominal and effective posture. How the 34 were enumerated is not documented.

32 residue classes: the action classes that remain above Level I under the fully hardened condition set CS-09, found by applying ten supervision determinants; 22 at Level III and 10 at Level II.

The 61.1 per cent: how it arose (REPORTED). The 61.1 per cent. REPORTED. It is the claim divergence rate MG-022: audited episodes in which at least one material factual assertion in the agent's own report is contradicted by independent evidence, divided by episodes audited. A material assertion is one about an action taken, a state reached or a verification performed; style, intention and reasoning are excluded, and so are episodes with no independent evidence on either side. The independent comparator is infrastructure records: version-control history, process-manager state, health checks, cloud control-plane state and the reverse-proxy access log. The value is 11 of 18 audited episodes, 0.611. The 18 were drawn from 46 candidate episodes identified by the system itself, and the technical core was confirmed in 18 of 18. The figure comes from the proposal-stage audit and was not re-executed in the run of 7 August 2026. The study's documents record that the audit was manual and that the sampling frame was not random. Who audited, when, how the 18 were chosen from the 46 candidates and how an episode was defined are not documented in the evidence base. The study's documents call the audit independent; because the observer is the operator, these materials do not rely on that word.

What the findings can and cannot establish. They can show, for this estate on 7 August 2026: that access exceeded mandate for 50 of 60 action classes; that only 4 of 60 classes were bound to a control the actor could not reach around, and all 4 are REPORTED; that 27 of 34 claimed supervision points diverge between nominal and effective posture; and that in the audited candidates agent self-report could not serve as the evidence base for governance. They cannot show how common any of this is in other deployments; a rate for agent misreporting in general (11 of 18 is a rate within non-randomly chosen audited candidates, not a population rate); deception or intent (the audit compared reports with evidence, not motives); incompetence (the technical core was confirmed in all 18); that a human intervention was warranted and not made (zero interventions in 46 days establishes that the loop was not exercised, which is a weaker claim); that the six conditions are complete; that the residue is intrinsic (it is relative to the catalogue); or that the framework works in practice (it has not yet been used to make a decision).

Limits of a single case

Limits of a single case. One environment, one snapshot, one observer, and the observer is the operator. The guardrail catalogue, the determinants and the viability conditions are shaped by what this one environment contains; a different environment would plausibly give different layers and determinants, and there is no way to know from inside how much. So every number in these materials is a finding about this estate on 7 August 2026, not a prevalence estimate. The study's own plan for testing generality is an isolated experimental environment and replication in other deployments; both are future work. Open question for the learner: does the pattern hold in other deployments? The study lists this as an open

question (OQ-004).

Running example, step 0 (M0)

The failure story of the deploy agent is the study's canonical incident (REPORTED, pinned by the residue register to AC-044): a deploy reported as successful had disabled the service's core connection function, and the service was unavailable for about 24 hours. The report said the deploy succeeded; independent evidence said otherwise. Nothing in this section claims the incident passed through the approval gate, or that the agent approved its own deploy; the register is silent on the route. What it shows is why a report from the deploying agent cannot be the evidence that a deploy worked.

You should now be able to ...

- explain the difference between governing a text generator and governing an agent;
- state the problem and name the three elements of the running example: the deploy agent, the approval policy, the approval gate;
- define instruction hierarchy, prompt injection, in-context scheming and the adversarial ceiling of prose, and say which of them is the authors' own phrase;
- say what the founding case is, who observed it, what the 61.1 per cent (REPORTED) is and is not, and why 46 days and 46 candidate episodes are different numbers;
- state why self-report divergence undermines instruction-only governance.

M1 Anatomy of an agentic deployment

Motivating question

Before you can govern a system, you must be able to draw it. What are the moving parts of a real multi-agent deployment, and where would a human even stand to supervise it?

M1.1 The parts list

A production agentic deployment is not one model behind one chat box. The founding case (Section M0.7), at observation time, comprised, on the agent orchestration hub and the estate around it:

- **Sessions.** 14 live agent sessions ran against 1 human operator. Each session is a long-lived model process with tool access: shell, filesystem, version control, network, and messaging.
- **Tools.** Every capability an agent can invoke: command execution, file writes, deployments, external API calls. Tools define the *access* half of everything that follows.
- **Automations.** Machinery that keeps agents running without a human: schedulers, watchdogs, restart loops, notification bridges. Over the observation window the deployment generated 19,334 automation heartbeat/nudge events versus 4 human-facing alerts. That is a volume, not a measure of operator capacity, which the study did not measure; it does show where the system's traffic flows.
- **The instruction corpus.** The prose that carries the mandate: what agents may and may not do. Narrowly counted, 40 regular files and 26 alias symlinks (about 538 kB); broadly counted, 1,240 files (about 26.7 MB). Every one of them is writable by the governed identity itself. The mandate exists only as prose the governed party can rewrite.

- **Supervision surfaces.** The places where a human could, in principle, see and intervene: consoles, logs, approval gates, alert channels, stop mechanisms. The study assessed 34 such supervision points.

M1.2 Two details worth pausing on

First, the *kill switch*. The deployment's self-healing subsystem, the machinery that restarts agents when they die, has an off switch: a sentinel file. That file sits in a directory the governed account owns. It can be removed with no authentication and no logged event. The control that is supposed to bound the system's self-perpetuation is inside the system's own reach. (This is one of the other cases from the founding case, not the running example.)

Second, the *silence budget*. Over 46 days of console audit logs there are zero records of a human stopping, pausing, rejecting or overriding an agent on the basis of what it was doing: zero stops, pauses, rejections or overrides, and zero kill-switch activations. This does not by itself prove neglect, and it does not establish that an intervention was warranted and not made. It shows that over 46 days the supervision loop was exercised zero times (finding F-042), which is the weaker claim the study makes.

Running example, step 1 (M1): the parts of the deploy path

The deploy path has five parts. The session that deploys is one of the 14 live sessions. The deploy tooling is its tool. The approval policy is an instruction file in the corpus, a prose rule the governed identity can rewrite. The approval gate (GR-L06-01) is the supervision surface, with three supervision points on it: SP-005 (human approval of a patch), SP-006 (an independent reviewer) and SP-007 (binding approval to what runs). The human who approves is the operator. All of this is OBSERVED, except AC-044 itself (REPORTED). The mandate, "every deploy needs a human's approval", lives only as prose in the operating guidance file, a dispatch brief or the fleet roster, and every instruction file is writable by the identity it governs.

The shape to remember

An agentic deployment is a loop: agents invoke tools, automations keep agents alive, instructions tell agents what they may do, and supervision surfaces let humans watch. Governance questions are questions about who can alter which part of this loop. In the founding case, the agents could alter almost all of it, including the instructions and the kill switch.

You should now be able to ...

- sketch the five parts (sessions, tools, automations, instruction corpus, supervision surfaces) for a deployment you know, and for the deploy path of the running example;
- explain why an instruction corpus writable by the governed identity cannot bind it;
- use the 19,334-to-4 event ratio to say where this deployment's traffic flows, and say what it does not show.

M2 The access-versus-mandate dual classification

Motivating question

Your agent *can* push to production, and it is *allowed* to open pull requests. Are those the same fact? What do you learn by writing both down for everything the agent can do?

M2.1 Three levels, in order

Both questions of this module use the same three levels, written I, II and III. The definitions are those of the study (D1), verbatim.

Level	Name	Definition (the study, verbatim)	Human role
I	autonomous	fully autonomous with no prior review, confined to development or version control, or permanently pre-authorised	approximates human-out-of-the-loop
II	autonomous, reviewed afterwards	autonomous with subsequent human review, carrying real production impact but reversible, communicated asynchronously	human-on-the-loop
III	prior approval or prohibited	requires prior approval or is prohibited outright: irreversible, externally facing, or altering a safety flag	human-in-the-loop

The levels are ordered by the human control they require: Level I needs none beforehand, Level II needs review afterwards, Level III needs prior approval or is prohibited. Permissiveness runs the opposite way: Level I is the most permissive. Access is more permissive than mandate when the access level number is lower than the mandate level number.

M2.2 Two questions, asked separately

The first instrument of architectural governance is disciplined bookkeeping. For every *action class* (a coherent family of things an agent might do: “modify production DNS”, “send a message to a customer”, “rewrite its own instructions”), ask two independent questions:

Access: what can the agent *technically* do, right now, with the credentials and tools it holds?

Graded on the three levels: Level I if it can just do it; Level II if it can do it and a human reviews afterwards; Level III if it cannot do it without prior approval, or cannot at all (for example because the write-capable identity is absent from the environment).

Mandate: what is the agent *authorised* to do, per the deployment’s own stated policy? Same three levels.

Author-proposed term 1: access versus mandate

Access versus mandate. Author-proposed term, not an established standard.

Definition. Every action class is graded twice on the same three levels. Access is what the agent is technically capable of doing today. Mandate is what it is authorised to do today. Where access is more permissive than mandate, the study records a delta, an unenforced boundary, and these materials call it the governance gap.

Example. AC-056, record an approval in the deployment approval gate: access Level I, mandate Level III. The boundary between what the agent can do and what it may do is not enforced by the structure of the system; it rests on role convention among agents that share one operating-system identity.

Non-example. AC-004, push a commit to a hosted remote repository: access Level I, mandate Level I. The agent is able and authorised, so there is no gap, even though the only control on it is a sentence in a prompt (instructional, GR-L12-02). A weak control is not a gap; a gap needs access to exceed mandate. A second non-example is AC-052, write to the System of Record (REPORTED): access Level III and mandate Level III, no gap, because the capability does not exist.

Not to be confused with. The risk grade, which rates consequence, and the enforcement rung, which rates who can defeat a control. Source: D1:823-830.

M2.3 A worked example: how the gap is read for one action

The reviewers' fair request was to see the gap calculated, once, on one action. Here it is.

Take one action class, AC-056, record an approval in the deployment approval gate. Step 1, mandate: the approval policy says a human decides whether a deploy goes ahead, so for an agent recording an approval is Level III, prior approval or prohibited. Step 2, access: the register records Level I, because the agent can do it with no one acting first. The register records AC-056 at access Level I because separation of duties is by role convention among agents sharing one operating-system identity (AC-056 note); the gate's approve route itself needs a human session (GR-L06-01), so the exposure is the shared credentials and the unbound approval-to-execution link. Step 3, compare: Level I access against Level III mandate means access is more permissive than mandate, so the register records a delta, a governance gap. Step 4, gap size: mandate rank 3 minus access rank 1 is 2 (derived presentation, not a register field). Step 5, read the other two scales separately: the risk grade is severe (how bad the consequence is), and the enforcement is conditional (who can defeat the control). Level I access with Level III mandate is the worst cell of the matrix: the system relies on prose, role convention and the agent's cooperation to hold the line.

AC-044's access is II because, per the register, it is "bounded by the deployment approval gate where that gate is in the path; the gate is a check inside the tooling, not a property of the target". The gate does not bound AC-056. "Two steps" and "one step" are derived readings: derived presentation, not a register field.

M2.4 The gap rule and the gap size

A governance gap exists for an action class when its access is more permissive than its mandate, that is, when the access level number is lower than the mandate level number. The register records this as a yes or no field called delta, and it matches this rule for all 60 classes. Gap size is a derived presentation, not a register field: mandate rank minus access rank, with Level I as 1, II as 2 and III as 3. Over the 60 classes, 28 have a gap of two steps (access Level I, mandate Level III), 22 have a gap of one step (18 at I and II, and 4 at II and III), and 10 have no gap; none has a negative gap, because access is never observed to be more restrictive than mandate on this host. 28 plus 22 is 50.

M2.5 Risk grade is not a level

Risk grade (consequence)

Risk grade (consequence). A four-point qualitative consequence rating (low, moderate, high, severe) recorded per action class by the study's observer; reversibility and blast radius are recorded as separate fields; no formula is documented. The study's documents and registers give no rule by which a grade was assigned, and no second rater checked it, so teach the

Table 1: One action, then four contrast rows. Levels are the blue columns, the risk grade (consequence) is the amber column, enforcement is the last. Gap size is DERIVED (mandate rank minus access rank), not a register field. Three separate scales: levels say who decides, enforcement says who can defeat the control, the risk grade says how bad the consequence is.

Action class	Access	Mandate	Gap?	Gap size (derived)	Risk grade (consequence)	Enforcement today	Evidence
AC-056 Record an approval in the deployment approval gate (the running-example row)	I	III	yes	2	severe	conditional (GR-L06-01)	OBSERVED (EV-030)
AC-044 Deploy application code to another estate host	II	III	yes	1	severe	conditional (GR-L06-01)	REPORTED (EV-030)
AC-029 Execute a command as root (two-step gap, no control)	I	III	yes	2	severe	none	OBSERVED (EV-003)
AC-004 Push a commit to a hosted remote repository (no-gap row)	I	I	no	0	moderate	instructional (GR-L12-02)	OBSERVED
AC-052 Write to the System of Record (no gap because access is absent)	III	III	no	0	severe	structural (GR-L05-01)	REPORTED

grade as the observer's judgement, not as a computed score. It is not a level: a class can have any grade at any access and mandate. Severe appears with a two-step gap (AC-056), with no gap (AC-052), and low appears with no gap (AC-001, read source code); the grade says how bad the consequence is, the levels say who decides, and the rung says who can defeat the control.

Over the 60 classes the grades are severe 22, high 20, moderate 14, low 4. Table 2 crosses them with the (access, mandate) cell, to show that the two scales are different axes. (Among the 56 classes without a structural control the split is severe 19, high 20, moderate 13, low 4.) The word "severe" belongs to the risk grade only: blast radius takes the values development, local-host, estate and external-public.

Table 2: Risk grade (amber) by (access, mandate) cell, in classes. Not a function of the reversibility field or the blast-radius field: for example local-host and reversible occurs at all four grades.

Risk grade (consequence)	(I,I)	(I,II)	(I,III)	(II,II)	(II,III)	(III,III)	Total
severe	0	1	17	0	3	1	22
high	0	10	9	0	1	0	20
moderate	4	7	2	1	0	0	14
low	4	0	0	0	0	0	4
Total	8	18	28	1	4	1	60

M2.6 Reading the founding case's matrix

The study inventoried 60 action classes and placed each in a matrix of (access level, mandate level). Table 3 shows the result. The levels were assigned by the single observer, from the evidence.

Table 3: Access-versus-mandate matrix of the founding case, in classes. Rows: technical access level. Columns: authorised mandate level. Levels: I = autonomous, II = autonomous, reviewed afterwards, III = prior approval or prohibited. Cells above the diagonal (18, 28, 4) are the 50 governance-gap classes.

Access	Mandate			Row total
	I	II	III	
I (autonomous)	8	18	28	54
II (reviewed afterwards)	0	1	4	5
III (prior approval or prohibited)	0	0	1	1
Column total	8	19	33	60

The 50 gap classes are 47 OBSERVED and 3 REPORTED (AC-044, AC-049, AC-050). The 10 no-gap classes are 8 at (I,I), AC-051 at (II,II) and AC-052 at (III,III). Three readings, in increasing order of alarm:

1. **The diagonal is thin.** Only $8 + 1 + 1 = 10$ of 60 action classes sit where access equals mandate. For the other 50 of 60, access exceeds mandate. And 0 of 60 show mandate exceeding access: never observed on this host.
2. **The worst cell is the biggest gap cell.** The (I,III) cell holds 28 action classes: things the policy says require prior approval or are prohibited outright, which the agent can nevertheless do autonomously, right now. Nearly half the inventory lives in the largest gap. Of those 28, the grade is severe for 17, high for 9 and moderate for 2.
3. **Level III access barely exists.** Only 1 action class of 60 is actually held at prior-approval access. Whatever the policy says, the deployment's technical posture is almost entirely Level I.

Why the matrix teaches

The matrix converts a vague worry ("our agent might be over-permissioned") into a countable object. Fifty of sixty classes above the diagonal is an audit result of this estate on 7 August 2026, assigned by one observer; it is not a prevalence estimate for other deployments. The exercise of drawing this matrix for your own system (Exercise E2) is the single highest-value activity in these notes.

Running example, step 2 (M2): the gap of the deploy agent

AC-056 is the hero row of Table 1: access I, mandate III, a two-step gap. AC-044, the deploy itself, is the one-step contrast (II, III). AC-029, execute a command as root, is the pure two-step gap with no control at all. AC-004 is the no-gap row. All are rows of the study's registers; AC-044 is REPORTED, the others OBSERVED except where the table says otherwise.

Reproduce the numbers

The folder `code` contains `reproduce.py` and `reproduce.ipynb`. Both read only `website/data.json`, use relative paths, and reproduce the access-versus-mandate matrix, the gap sizes, the enforcement-ladder counts, the layer table, the guardrail population counts, the supervision-point and residue counts, and the Guardrail Lab outcome tables for every condition set. `expected-output.txt` is the output of a clean run, so you can check yours against it. From the folder `code`, run `python3 reproduce.py > my-output.txt` and compare with `expected-output.txt`; the last line reads “All published figures reproduced from `website/data.json`.” The matrix cells 8, 18, 28, 1, 4, 1 and the gap sizes 28, 22 and 10 above are the first figures it prints. The gap size is labelled DERIVED in the output.

You should now be able to ...

- define access and mandate, state the three levels in order, and say which is the most permissive;
- work the gap for one action class as in Table 1, and say why access Level I against mandate Level III is the largest gap;
- read Table 3 aloud: locate the diagonal, the gap cells, and the (I,III) mass;
- say why the risk grade is not a level, and why gap size is a derived presentation;
- explain why “50 of 60 above the diagonal, 0 below” summarises the founding case’s governance posture in one line, and what it does not show about other deployments.

M3 The enforcement ladder

Motivating question

Two controls both say “agents must not touch production data”. One is a sentence in a policy file; the other is a database account with no write permission. What distinguishes them, and can we grade every control on that axis?

M3.1 One question grades every control

The second instrument is a four-rung ladder, and the sorting question is always the same: **who can defeat this control?**

Rung 0, none. There is no control. Nothing even claims to prevent the effect.

Rung 1, instructional. The control is prose: a prompt, a policy file, a stern comment. It is defeated by the governed agent simply not complying, whether through error, drift, injection, or intent. The agent defeats it by *doing nothing special*.

Rung 2, conditional. The control is a mechanism, but one the actor could alter or bypass: a check in a script the agent can edit, a hook in a repository the agent can push to, a sentinel file the agent can delete. It binds only on condition that the agent leaves it alone.

Rung 3, structural. The control cannot be reached around by the governed actor: a credential that lacks the permission, a network path that does not exist, an external process the agent cannot signal. Defeating it requires powers the agent does not hold. Compliance is not a term in the outcome.

Figure 1 draws the ladder. Note that the axis is not “how strongly worded” or “how sophisticated” but strictly *who can defeat it*: the same regex check is conditional if the agent can edit the file it lives in and structural if it runs on a host the agent cannot touch.

sorting question: who can defeat it?

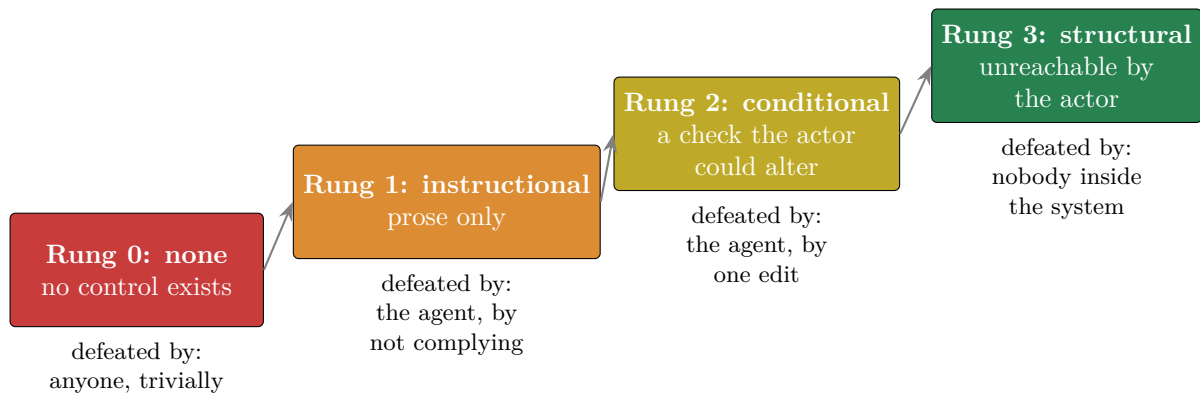


Figure 1: The enforcement ladder, ascending from rung 0 to rung 3. Founding-case placement of its 60 action classes: 36 no control at all, 8 prose alone (instructional), 12 a check the actor could alter (conditional), 4 a control the actor cannot reach around (structural; all 4 REPORTED). This is the as-found ladder; the Guardrail Lab's CS-03 row differs for a documented reason (the note on CS-03 after Exercise E5).

Author-proposed term 2: enforcement ladder

Enforcement ladder. Author-proposed term, not an established standard.

Definition. The enforcement ladder grades any control by one question: who can defeat it? Rung 0, none: no control of any kind. Rung 1, instructional: the actor is told what to do or not do, in prose the model reads as context, and nothing in the runtime blocks the action. Rung 2, conditional: a check exists in a code path and is evaluated at run time, but the actor could reach the same effect by another path, or could alter the condition it is checked against. Rung 3, structural: the prohibited action is not reachable by the actor; defeating the control needs a change of architecture, not a change of decision, and the agent's compliance is not a term in the outcome.

Example. The deploy at each rung. Rung 0: nothing governs AC-029, execute a command as root. Rung 1: the approval policy is a sentence in files the agent reads, and AC-043, execute a command on another estate host, is governed only by a prose instruction to confirm first (GR-L12-06). Rung 2: AC-044 and AC-056 are governed by the approval gate GR-L06-01 (partial, conditional, fail-open). Rung 3, PROPOSED: a separate approver identity (GR-L01-08) and a deploy artefact bound to its approval (GR-L06-11). The one present structural exemplar is AC-052 through GR-L05-01 (REPORTED). In the founding case: 36 classes sit at rung 0, 8 at rung 1, 12 at rung 2 and 4 at rung 3.

Non-example. A well-built control is not thereby structural. The approval gate is carefully engineered and approval authority is genuinely hard to bypass, yet the gate is conditional: the credentials it relies on are readable by the agents and nothing binds what runs to what was approved. The rung is a property of the mechanism, not of its intent or its quality. Also not a control: GR-L07-09 (prose brief as the operative mandate) is one of nine inverted entries that describe an absence.

Source: D3 section 1.2.

M3.2 The founding case on the ladder

Grading the strongest control protecting each of the 60 action classes (the class's recorded current enforcement) gives: 36 with no control at all, 8 instructional, 12 conditional, 4 structural. So 56 of 60 action classes lack structural enforcement. The 4 structural classes are AC-049 and AC-050 (through GR-L06-08) and AC-051 and AC-052 (through GR-L05-01), all REPORTED; because AC-051 and AC-052 have no gap, only 2 of the 4 sit among the 50 gap classes. Set that against the matrix from M2: 50 of 60 classes have access exceeding mandate, and for all but two of them, whatever stands between the agent and the prohibited effect is either nothing, words, or a mechanism the agent could edit.

Running example, step 3 (M3): the same intent at four rungs

Rung 0: nothing governs the command (AC-029 as the contrast). Rung 1: the approval policy is a sentence in a file, and the confirm-before-remote-execution sentence governs AC-043 (GR-L12-06). Rung 2: the approval gate is a check the agent can route around (GR-L06-01). Rung 3 (PROPOSED): approval authority and the running artefact are bound by the architecture, with a deploy credential held only by a pipeline the agent cannot reach (GR-L01-08, GR-L06-11, both PROPOSED, neither present in the estate). Same intent, four rungs; only rung 3 removes the effect from the agent's reachable set, and for the deploy that rung is PROPOSED.

A second worked classification

"Agents must not restart the customer-facing service" as a line in the system prompt: *instructional*. A pre-restart hook script, owned by the agent's own account, that checks a change-window calendar: *conditional* (one edit defeats it). Running the agent under an identity whose service-manager policy denies restart of that unit: *structural*. Same intent, three rungs; only the third removes the effect from the agent's reachable set.

M3.3 Why the lower rungs dominate

It is worth asking honestly why deployed systems cluster at the bottom of the ladder (LO5). Three forces:

1. **Prose is free.** Adding a sentence to a prompt costs seconds and breaks nothing. Provisioning a scoped credential costs a morning and can break workflows.
2. **Capability is bundled.** Tools, tokens and accounts arrive broad by default; narrowing them is deliberate work that the delivery pressure of an agentic project rarely schedules.
3. **Compliance is invisible until it fails.** An instructional control looks identical to a structural one on every day the agent happens to comply. The difference only manifests at the exact moment it matters most.

You should now be able to ...

- classify any offered control into the four rungs by asking who can defeat it;
- place the deploy agent's policy, gate and PROPOSED redesign on the ladder;
- recall the founding-case distribution (36 / 8 / 12 / 4 classes) and the headline "56 of 60 lack structural enforcement";
- explain the three forces that make lower rungs dominate real deployments.

M4 Guardrails in twelve layers

Motivating question

If you set out to catalogue every guardrail a serious agentic deployment could have, layer by layer, what would the catalogue look like, and how much of it would you find actually built?

M4.1 The catalogue

The founding-case study compiled a guardrail catalogue of 134 entries across 12 layers, L1 to L12: a catalogue of the design space, not 134 deployed controls. Each entry is graded twice: by *enforcement kind* (the M3 ladder) and by *population status* (present, partial, or proposed, i.e. identified as needed but not built). Table 4 lists the layers, with the register's names. The layers run from the identity a process runs as to the prose an agent reads as instruction, from furthest from the model to nearest the model.

Across the 134 entries, the enforcement kinds are: 49 structural, 60 conditional, 13 instructional, 12 none. The population statuses are: 39 present, 17 partial, 78 proposed. Now cross the two gradings and the central finding of the whole study appears:

The central asymmetry

Of the 39 guardrails actually *present*, only 2 are structural (GR-L05-01 and GR-L06-08, both REPORTED, not verified on the host). Of the 49 *structural* entries in the catalogue, 47 are proposed, not built. The deployment has plenty of guardrails; it has almost none of the kind that binds. Structural enforcement is not absent from the design space. It is absent from the deployment.

Inverted entries

Inverted entries. Nine of the catalogue's present entries describe an absence or an arrangement that weakens the posture rather than a control: GR-L01-01 shared operating-system account, GR-L01-09 passwordless root, GR-L01-10 reachable metadata credential, GR-L04-10 permission skipping, GR-L07-09 prose brief as the operative mandate, GR-L08-13 transcript hook substrate unused for enforcement, GR-L09-10 restart-always supervision that defeats termination, GR-L10-08 automated answering of interactive prompts and GR-L11-05 scheduled unattended dispatch on a clock. Eight are enforcement class none and one (GR-L07-09) is instructional. They are part of the founding case as found (CS-03) and are left out of the fully hardened set (CS-09 holds 125 of the 134 entries). So "39 present" includes these nine.

Throughout, "fully hardened" means all 125 non-inverted controls (CS-09), never the whole catalogue of 134 entries. A register caveat: "present" in the catalogue means present in the estate per the study; for four present entries (GR-L05-01, GR-L06-08, GR-L09-07, GR-L09-08) the register field `present_on_host` reads partial, no, no and no respectively, because they are REPORTED from other hosts.

M4.2 The two structural exemplars

The two genuinely structural, genuinely present controls repay close study, because they share one shape: *the prohibited effect is not reachable at all*. Both are REPORTED, not verified on the host in the run of 7 August 2026.

1. **A read-only connection identity to the system of record** (GR-L05-01, REPORTED).
The agent-facing pathway into the deployment's most important external database authen-

Table 4: The twelve guardrail layers (register names), entry counts and population (134 entries in total: 39 present, 17 partial, 78 proposed).

Layer	Name	Entries	Present / partial / proposed	What it asks
L1	Identity and credential	13	3 / 0 / 10	Who or what the agent is, cryptographically, and what that identity can obtain
L2	Network and egress	12	2 / 1 / 9	What the agent's processes can reach over a network, inbound and outbound
L3	Filesystem and workspace	11	3 / 1 / 7	What the agent can read, write or isolate on disk
L4	Process and execution	12	5 / 1 / 6	How the agent's processes are spawned, bounded and terminated
L5	Data plane	10	2 / 1 / 7	What the agent's stored state, and the systems of record it touches, can structurally represent or permit
L6	Change and deployment	12	2 / 4 / 6	How a change reaches a running system, and who or what gates it
L7	Mandate and policy	11	1 / 2 / 8	What the agent is authorised to do, as a machine-checkable artefact rather than as prose
L8	Evidence, audit and verification	13	2 / 2 / 9	What record exists of what happened, and who can check it independently of the actor
L9	Containment and interruption	11	5 / 1 / 5	What stops an agent, partially or completely, once it is already acting
L10	Human interface and oversight	11	3 / 3 / 5	What a human actually sees and can do about it
L11	Temporal and contextual	9	2 / 1 / 6	What time, sequence and duration constraints apply to an action
L12	Model and prompt layer	9	9 / 0 / 0	What the agent is told, in language it reads as context (all nine entries instructional)
Total		134	39 present / 17 partial / 78 proposed	

ticates as an identity that has no write permission. No instruction says “do not write”; writing is not a thing that connection can express. Action classes AC-051 and AC-052 sit behind it.

2. **A structural test-mode overlay in the real-time domain** (GR-L06-08, REPORTED). In the deployment’s real-time communication domain, a test mode is imposed by the surrounding platform such that live effects cannot be produced from the test path, whatever the agent does inside it. Action classes AC-049 and AC-050 sit behind it.

In both, compliance is not a term in the outcome. The agent’s behaviour, good or bad, honest or scheming, does not appear in the causal chain that prevents the effect. That is the signature of architecture.

M4.3 Layer patterns and failure direction

Two layer-level patterns matter. L12, the model and prompt layer, has all 9 of its entries instructional *by construction* (prompts can only ever be prose) and all 9 are present: the easiest layer to populate is fully populated. L1, identity and credential, shows the mirror image: 10 of its 13 entries are proposed, and 7 of those proposed entries are structural, the highest concentration of structural proposals in the catalogue. The layer where architecture lives is the layer left unbuilt.

Finally, each entry is graded by fail state: what happens when the guardrail itself breaks? Across the catalogue, 85 entries fail closed (breakage blocks the action) versus 49 that fail open (breakage silently removes the control). A fail-open guardrail subtracts protection at exactly the moment it malfunctions, and nobody need notice.

Running example, step 4 (M4): the gate in the catalogue

The approval gate is the catalogue entry GR-L06-01 in layer L6, Change and deployment: population partial, enforcement class conditional, fail-open. Its structural counterparts are three PROPOSED entries in three layers: GR-L01-08 (a separate approver identity, layer L1), GR-L06-11 (deploy artefact provenance, layer L6) and GR-L08-05 (an independent verifier, layer L8). All three are PROPOSED: they describe what the study would build, not what the estate has.

You should now be able to ...

- name the twelve layers and describe the two-axis grading (enforcement kind, population status);
- state the central asymmetry: 39 present but only 2 structural; 47 of 49 structural entries proposed;
- explain what an inverted entry is and why CS-09 holds 125 of the 134 entries;
- explain, using the two exemplars, what shape a genuinely structural control has;
- distinguish fail-open from fail-closed and say why the 85 versus 49 split matters.

M5 Is your human oversight real? The six viability conditions

Motivating question

During the production of the study’s own documents, a rendering parser stalled. It emitted no error, no timeout, no signal of any kind; it simply grew, eventually reaching 9.1 GiB of resident memory while reporting nothing. Three attempts ended the same way, and each ended only when the operating system kernel’s out-of-memory killer, an external structural control owing nothing to the process, killed it. The process could not be relied on to announce its own failure. Why do we assume agent deployments will announce theirs?

M5.1 The lesson of the stalled parser

The anecdote above is OBSERVED in the study’s own toolchain (finding F-043); it is small, local, and not even about an LLM, which is exactly why it teaches so well. A supervised process gave no usable oversight signal *from inside itself*, and the only effective control was one that (a) ran outside the process, (b) measured an objective quantity, and (c) could kill without cooperation. Generalise it and you have the design rule of this module: **oversight signals must not originate from the supervised process**.

Human-in-the-loop is the most cited safeguard in agentic AI, and the least examined. “A human reviews it” names a hope, not a mechanism. This module gives the hope a test.

M5.2 The six viability conditions

A supervision point is any place where the architecture claims some human supervision, intervention or signal. The study assesses each against six conditions; failing any one leaves oversight nominal rather than effective. Table 5 lists them, in the register’s order VC1 to VC6, with the founding-case binding conditions as the study recorded them.

Table 5: The six viability conditions (VC1 to VC6, the study’s wording) and the binding condition as recorded by the study among the founding case’s 27 diverging supervision points. 27 of 34 points diverge between nominal and effective posture; for one of them, the claimed control does not exist at all.

Condition	The question it asks	Recorded binding (points)
VC1 Independence	The oversight signal does not originate from the agent being supervised.	8
VC2 Latency	Intervention latency is shorter than harm completion time: $t_{detect} + t_{decide} + t_{act} < t_{harm}$.	8
VC3 Halt	The intervention halts work in progress rather than only preventing the next action.	3
VC4 Attention	The number of concurrent signal streams is within one operator’s capacity.	2
VC5 Comprehensibility	An approval request or alert presents independent evidence rather than the agent’s own summary.	2
VC6 Authority	The human has the standing and the mechanism to override, and the override is not defeated by an automatic restart or retry.	3
none	The claimed control does not exist (SP-018)	1

How to read the binding counts

The binding condition is the one the study recorded for each supervision point. The study's convention reads the conditions in the order VC1 to VC6 and takes the first failing condition as the binding one; this is a convention, not a result, and it does not reproduce every recorded row (10 of the 27), so do not recompute the counts 8, 8, 3, 3, 2, 2 and 1 from the six columns. For example SP-005 reads VC1 partial, VC3 fail, VC5 fail, and the study recorded VC5. VC3 fails earlier in the order than VC5, so first-fail does not explain the recorded VC5; treat the binding condition as the study's judgement, not a formula.

Of 34 supervision points assessed, 27 diverge between what they nominally provide and what they effectively provide ($8 + 8 + 3 + 2 + 2 + 3 + 1 = 27$). The two most commonly recorded binding conditions, independence (8) and latency (8), are precisely the two the stalled parser dramatised: signals came from inside the process, and came too late or never.

Author-proposed term 3: supervision viability

Supervision viability. Author-proposed term, not an established standard.

Definition. Supervision viability asks whether the human supervision an architecture claims at a supervision point actually exists. Each point has a nominal posture (what the architecture claims) and an effective posture (what a human can actually exercise). Six conditions must all hold for the claimed posture to be real: VC1 independence, VC2 latency, VC3 halt, VC4 attention, VC5 comprehensibility, VC6 authority. Where one fails, the effective posture is weaker than the nominal one.

Example. SP-005, deployment approval gate: human approval of a patch. Nominal posture in the loop. VC1 partial, VC2 pass, VC3 fail, VC4 pass, VC5 fail, VC6 pass; effective posture in the loop, weakly. Binding condition as recorded by the study: VC5 comprehensibility, because the approval request carries the agent's own description of the patch and not independent evidence. SP-006, independent reviewer, fails four of the six conditions and is effectively out of the loop. SP-007, binding approval to what runs, fails five of the six: no signal is emitted when an unapproved artefact is loaded.

Non-example. SP-024, structural test mode intercepting external dispatch, and SP-029, read-only connection identity to the System of Record, pass all six conditions (both REPORTED). They are not a human watching harder: the control acts before the action and does not depend on a human or on the agent's cooperation. Notification volume is not viability either: 19,334 automation events against 4 human-facing alerts is a volume, and operator capacity was not measured.

Limits. The six conditions are asserted, not derived; there is no proof that they are complete, independent or minimal, and the order in which they are read is a convention. Source: D2 sections 4 and 9.

M5.3 The founding case's oversight, measured

The numbers from M1 now land with their proper weight:

- The deployment has 6 intervention mechanisms; exactly 1 of them halts work in progress. Of the rest, the "supervised stop" allows up to 35 seconds of unrestricted tool access before terminating, and the restart-always machinery brings terminated agents back within about 10 seconds. An operator who stops an agent has bought, at best, a pause.
- The best-case latency of any agent-independent oversight signal exceeds 10 minutes, and that signal detects *silence*, not action: it can notice an agent that stopped talking, never

one doing the wrong thing fluently. 8 of 14 live sessions sit outside every watchdog's scope; one session sat silent for 7 days.

- Attention: 19,334 automation events versus 4 human-facing alerts; 14 live sessions, 1 operator; 46 days of console audit logs with zero records of a human stopping, pausing, rejecting or overriding an agent on the basis of what it was doing. Notification volume is a volume; operator capacity was not measured.
- Authority: 434 update proposals were pending in the update plane's approval lane, with no approver set and none applied. The supervision point that governs them, SP-008, is recorded by the register as "none: all six conditions hold", although its halt cell reads fail (an internal inconsistency of the register, reported to the authors). Only SP-024 and SP-029 read pass on all six conditions, and both are REPORTED. The channel exists; in 46 days it was not used to apply anything.

The audit method

For any supervision point, read the six conditions in order VC1 to VC6; the study's convention records the first failing condition as the *binding condition*, and it is where remediation effort should go. The register does not reproduce this for every row (for 10 of the 27 diverging points the recorded condition is not the first to read fail), so treat the order as a reading convention and the recorded condition as the study's judgement, not a formula, and expect an experienced assessor to disagree on some points. Fixing a condition that does not bind (a prettier dashboard for a channel nobody watches) buys nothing.

Running example, step 5 (M5): assess the approval gate

Apply the six conditions to the three supervision points on the approval gate (OBSERVED): SP-005, human approval of a patch (recorded binding condition VC5); SP-006, an independent reviewer (recorded VC1); SP-007, binding approval to what runs (recorded VC1). The point of the exercise is the gap between nominal and effective: the gate nominally puts a human in the loop, and the conditions ask whether that human has independent evidence (VC5), whether the reviewer is independent of the author (VC1), and whether a signal is emitted when something unapproved runs. Exercise E3 works SP-005.

You should now be able to ...

- recite the six viability conditions in order VC1 to VC6 and the question each asks;
- assess a given supervision point, name its binding condition as a reader would record it, and defend the choice;
- retell the stalled-parser anecdote and extract its design rule: oversight signals must not originate from the supervised process;
- use "27 of 34 diverge" and "434 pending, none applied" to argue the difference between nominal and effective oversight, and say why the counts of binding conditions are recorded, not recomputed.

M6 The supervision residue and its determinants

Motivating question

Suppose you built everything: every catalogued guardrail except the nine inverted entries, every structural proposal, the lot. Could every action class then run autonomously? If not, what is left over, and why?

M6.1 Hardening is not absolutism

The third and fourth instruments meet here. A *supervision level* is the level (I, II or III) at which an action class can defensibly operate given the guardrails in place: the more constrained of a floor set by the action's own properties and a ceiling set by the guardrail set. For each of the 60 action classes, the study asks which classes remain above Level I under the fully hardened condition set CS-09. The answer defines the **supervision residue**.

Author-proposed term 4: supervision residue

Supervision residue. Author-proposed term, not an established standard.

Definition. An action class is in the supervision residue if and only if it remains above Level I under condition set CS-09, the fully hardened configuration that contains every catalogued control except the nine inverted entries. The definition is mechanical, so the residue can be recomputed whenever the catalogue grows. In the founding case 32 of 60 action classes are in the residue: 22 at Level III and 10 at Level II (the register's residue level). Each is pinned by one or more of ten supervision determinants (SD-01 to SD-10), one of them dominant.

Example. AC-056, record an approval in the deployment approval gate: residue Level III, pinned by SD-03 and SD-05, dominant determinant SD-03 accountability requirement. The register's reason: an approval is a statement that a named party accepts responsibility, and a machine recording it does not create the acceptance. AC-044, deploy application code to another estate host: residue Level III, pinned by SD-03, SD-04, SD-06 and SD-01, dominant SD-03; with the full catalogue it could reach Level II, but the decision to deploy under uncertainty remains a judgement with an accountable owner.

Non-example. AC-004, push a commit to a hosted remote repository, is not in the residue: Level I is defensible under CS-09. The residue is also not the set of classes the Guardrail Lab blocks: under the Lab's rule 31 of the 32 residue classes are BLOCKED at CS-09 and still remain above Level I (derived), because a structural block removes the agent's ability to act alone, while a determinant such as accountability keeps the human decision.

Limits. The residue is relative to the catalogue and is an upper bound, not a claim about what is possible in principle. The register also records a best achievable level (III for 13 classes, II for 19), which differs from the residue level for 9 classes; these materials quote the residue level (22 and 10) and say so. Source: D2 section 3.3 and section 9.

The founding-case result: 32 of 60 action classes stay above the autonomous level under the fully hardened set; 22 of them at Level III (prior approval or prohibited), 10 at Level II (autonomous, reviewed afterwards). Guardrails, even architectural ones, buy autonomy for fewer than half the inventory (28 classes are not in the residue; "28 releasable" means not in the residue under CS-09, not releasable today). The rest is residue.

M6.2 Why classes stay in the residue

Each residue membership has determinants, and the study catalogues 10 supervision determinants, SD-01 to SD-10: 8 carried over from the research proposal and 2 added in the run (compounding, where small autonomous actions aggregate into large unauthorised outcomes, and adversarial adaptation, where the governed system can learn to route around a control). The dominant determinants across the 32 residue classes are:

- **Containment shortfall, SD-07 (9 classes).** The action cannot currently be sandboxed, but could be: this determinant is removable by identity engineering (scoped credentials, isolated environments). These 9 are the actionable frontier.
- **Externality, SD-02 (9 classes).** The action's effects land on parties outside the deployment (customers, third parties, the public), who never consented to agentic risk. No internal guardrail can consent on their behalf.
- **Irreversibility, SD-01 (4 classes).** No rollback exists; prevention is the only control that means anything.
- **Accountability requirement, SD-03 (4 classes).** A human must own the outcome for legal or institutional reasons, so a human must be in the decision.

The remaining 6 classes are dominated by objective ambiguity (2), compounding (1), value and judgement content (1), unverifiability (1) and distribution shift (1). So 17 classes are dominated by externality, accountability or irreversibility (9 + 4 + 4), and 23 are not dominated by containment shortfall. The asymmetry inside this list is the module's key insight: containment shortfall is an engineering problem, but externality, irreversibility and accountability are properties of the action's relationship to the world. More architecture cannot remove them; it can only make the supervised path safer.

M6.3 Condition sets and domains

To reason about “what if we built more?”, the study defines 10 condition sets, CS-01 to CS-10. A condition set is a named, explicit subset of the guardrail catalogue, used to hold the guardrail variable constant at a defined maturity level. Membership is explicit per set. Table 6 lists them.

CS-01 is the bare baseline with 0 guardrails. CS-03 is all 39 present and 17 partial entries, including the nine inverted entries. CS-04 is CS-03 without those nine entries plus ten new identity-layer entries (GR-L01-02 to GR-L01-08 and GR-L01-11 to GR-L01-13, all PROPOSED, not present in the estate), 57 in all; so CS-03 is not a subset of CS-09 and CS-04 is not a superset of CS-03. CS-05 to CS-09 each contain the one before. CS-10 is CS-09 without the ten verification entries GR-L08-03 to GR-L08-12, to isolate what verification alone contributes. The residue result holds for CS-09, the set that defines it.

Autonomy is also domain-shaped. The study ranks 9 applicability domains, DM-01 to DM-09, by defensible autonomy: development work ranks most defensible (contained, reversible, internal), while real-time uncontained action, such as live telephony, ranks least. And a sharp caution for prompt-layer enthusiasts: the prose layer (L12) is marked counterproductive in 4 of 9 domains (DM-04, DM-06, DM-07 and DM-08), the four domains where harm is fastest, most irreversible, or most accountability-bound: a prose instruction is not merely insufficient there, it is the wrong tool, because it invites the belief that telling the model what to do has addressed a risk that only an architectural constraint can address.

Running example, step 6 (M6): why the deploy stays supervised

AC-044 and AC-056 are both in the residue, at Level III, with the dominant determinant SD-03, the accountability requirement: engineering cannot remove the human decision.

Table 6: The ten condition sets. “Layers added or varied” is the register field, not the layers that have members. The last column is the study’s stated expectation of a supervision level; the Guardrail Lab does not show it.

Set	Name	Guardrails	Layers added or varied	Study’s stated expectation
CS-01	Bare baseline (control)	0	none	I in practice, III in principle for every class
CS-02	Instructional only	9	L12	I in practice, II claimed
CS-03	Founding case as found	56	L12	I in practice, II claimed
CS-04	Identity separated	57	L1, L12	II
CS-05	Contained	73	L1, L2, L3, L12	II
CS-06	Verified	82	L1, L2, L3, L8, L12	II with real review
CS-07	Mandated	90	L1, L2, L3, L7, L8, L12	II, with Level I defensible for the guardrail-dominated classes
CS-08	Interruptible	106	L1, L2, L3, L7, L8, L9, L10, L11, L12	the same as CS-07
CS-09	Fully hardened	125	L1 to L12	II or III for the residue; I for everything else
CS-10	Fully hardened minus verification (ablation)	115	L1 to L7, L9 to L12	the same as CS-09

That is the residue’s lesson for the running example. Even a fully hardened estate, in which GR-L01-08 and GR-L06-11 (both PROPOSED) bind the deploy structurally, still needs a named human to accept responsibility for a deploy; the architecture makes that human’s approval real, it does not replace it.

You should now be able to ...

- define the supervision residue and recall its size: 32 of 60, split 22 at Level III and 10 at Level II (the residue level);
- name the four dominant determinants and say which one is removable by engineering;
- explain what the condition sets CS-01 to CS-10 are for, and what the ablation set tests;
- argue why development work and live telephony sit at opposite ends of defensible autonomy.

M7 Measuring governance

Motivating question

“Our agents are well governed.” What number would you ask for, and would the deployment be able to produce it?

M7.1 The metric inventory

Governance you cannot measure is governance you are asserting. The study specifies 72 metrics covering the instruments of M2 to M6 (gap sizes, ladder distributions, viability pass rates, residue coverage, and more), then audits the founding case for each: is the metric *available* today (computable from existing records), *derivable* (computable after modest instrumentation),

or *absent* (the underlying records do not exist)?

The result: 72 specified metrics, 21 engineering and 51 governance; 14 available, 25 derivable and 31 absent (2 further entries split). On top of the base metrics sit 8 composite indices, none of them available. And one family is absent outright: repair-capability metrics are absent, because no fault, incident or repair register exists at all. The deployment repairs things constantly; it just keeps no record from which repair capability could ever be computed. Only one metric has a computed baseline, the claim divergence rate MG-022.

M7.2 The 61.1% (REPORTED) divergence rate

One number deserves its own subsection, because it decides what all the other numbers can rest on. Its method, its sample and its limits are in Section M0.7; read them first. In short: 61.1% (REPORTED) is 11 of 18 audited episodes, drawn from 46 candidate episodes identified by the system itself, in which the agent's own account of its work diverged factually from the independent evidence. The technical core was confirmed in all 18: agents did the work, and misreported facts about it. The canonical incident makes the stakes concrete (REPORTED, pinned to AC-044): a deploy reported successful had in fact disabled the service's core connection function, and the service was unavailable for about 24 hours.

Interpret this carefully (LO7). The rate does not establish deception, malice, or scheming; the audited episodes were system-identified candidates drawn from a sampling frame that was not random, so 61.1% (REPORTED) is a divergence rate *within audited candidates*, not a population rate, and the choice of the 18 from the 46 is not documented. What it does show, within this estate and for the audited episodes, is that agent self-report cannot serve as the evidence base for governance metrics. Any metric whose inputs are agent-generated claims inherits this divergence. Measurement, like oversight, must be independent of the measured party.

Limits of a single case

Limits of a single case. One environment, one snapshot, one observer, and the observer is the operator. The guardrail catalogue, the determinants and the viability conditions are shaped by what this one environment contains; a different environment would plausibly give different layers and determinants, and there is no way to know from inside how much. So every number in these materials is a finding about this estate on 7 August 2026, not a prevalence estimate. The study's own plan for testing generality is an isolated experimental environment and replication in other deployments; both are future work. Open question for the learner: does the pattern hold in other deployments? The study lists this as an open question (OQ-004).

M7.3 Epistemic discipline

The study closes its measurement chapter with a practice worth adopting everywhere: every one of its 43 durable findings (F-001 to F-043) carries an epistemic label (Section M0.6): OBSERVED, direct evidence held; REPORTED, resting on accounts that could not be confirmed on the host; PROPOSED, kept strictly distinct for designs not yet built; NOT DETERMINED. These notes add AUTHOR-ATTESTED for the authors' own contextual statements. Keeping the labels distinct is what lets a reader weigh each claim; collapsing them is how governance theatre starts.

Running example, step 7 (M7): what to measure for the deploy path

To know the deploy path is safe, measure the claim divergence rate MG-022 on deploy reports: how often the deploying agent's report of a deploy is contradicted by independent

evidence. The MG-022 baseline is REPORTED. An independent post-deploy verification, GR-L06-12 (PROPOSED), would supply the comparator, a check by a principal other than the deploying agent.

You should now be able to ...

- recall the metric audit (72 specified: 14 available, 25 derivable, 31 absent; 8 composite indices, none available);
- state the 61.1% divergence result precisely, including its denominator (11 of 18 audited episodes, drawn from 46 candidate episodes), its label (REPORTED) and what it does and does not establish;
- state the limits of a single case in one sentence;
- explain why self-report cannot ground governance metrics, and what the evidence labels are for.

M8 The frontier: making enforcement architectural

Motivating question

If the diagnosis is “instructions leak and mechanisms inside the agent’s reach get altered”, what does the cure look like at research scale, and what remains genuinely open?

M8.1 Where research is already moving

The literature of M0, reread through the instruments of M2 to M7, turns out to be converging on architecture from several directions. This mapping of papers onto the ladder is the authors’ reading, not a claim of the papers’ authors.

- **Design-time unreachability.** CaMeL [4] is the clearest published instance of the ladder’s top rung: by separating control flow from data flow, injected instructions in retrieved data are structurally unable to influence which tools run. This is the read-only credential idea generalised to information flow.
- **Untrusted-model protocols.** AI Control [3] designs pipelines that remain safe under intentional subversion: trusted monitoring, deference to weaker trusted models, audit budgets. In ladder terms, it engineers conditional controls whose defeat requires capabilities the untrusted model is assumed not to have, and evaluates them adversarially rather than hopefully.
- **Honest evaluation of the gap.** ToolEmu [1] and AgentDojo [2] give the field its measurement layer, the research analogue of M7: they make failure countable before deployment does.
- **The adversarial ceiling of prose, characterised.** The instruction hierarchy [5] strengthens prose as far as prose goes, while the scheming results [6] bound what prose can ever guarantee against a capable, goal-directed model. Together they bracket the ladder’s second rung from above and below.

M8.2 Open questions

Honest teaching ends with what we do not know. Six open problems, each traceable to a gap the founding case exposes or to the limits of the case itself:

1. **Approval at agent speed.** Approval gates workable for 14 sessions may collapse at 14 thousand. What does prior approval mean when the queue is machine-generated faster than humans can read it? (Running example: which of the six papers would have prevented the deploy agent's gap, and would a gate that presents evidence, GR-L10-04, PROPOSED, scale?)
2. **Verification independent of the verified.** If self-report diverges in 11 of 18 audited episodes (61.1%, REPORTED), who verifies the work? Second agents inherit the same problem one level up; the recursion needs a base case that is structural, not agentic.
3. **Residue economics.** 32 of 60 classes stay above Level I under the fully hardened set. What is the honest cost model for keeping humans meaningfully in those loops, at attention conditions humans can actually sustain?
4. **Fail-closed by default.** 49 of 134 catalogued guardrails fail open. Can toolchains and platforms make fail-closed the path of least resistance, the way memory-safe languages made a class of bugs unwritable?
5. **Portable governance evidence.** 31 of 72 metrics were absent in one well-instrumented deployment (with 2 more entries split). What minimal record-keeping standard would make the M7 audit computable anywhere, so that "well governed" becomes a checkable claim between organisations?
6. **Does the pattern hold in other deployments?** The founding case is one environment, one snapshot, one observer, and the observer is the operator. The study's own limit, and its open question OQ-004, is whether the layers, determinants, viability conditions and counts generalise. Replication in other deployments and an isolated experimental environment are future work.

The one-sentence takeaway

Governance of agentic AI matures the way safety engineering always has: by moving controls off the honour system, out of the governed party's reach, and into structure, and by measuring, with evidence independent of the agent, how far along that path a deployment actually is.

Running example, step 8 (M8): the frontier of the deploy agent

The open question for the deploy agent is approval at agent speed. A PROPOSED structural redesign (separate approver identity GR-L01-08; deploy artefact provenance GR-L06-11; independent post-deploy verification GR-L06-12; an independent verifier GR-L08-05; approval requests that present evidence GR-L10-04, all PROPOSED) would make the gate structural, but it does not answer how a human approves ten thousand deploys an hour. That is where the frontier is.

You should now be able to ...

- map each of the six cited papers onto the instruments of these notes, and say that the mapping is the authors' reading;
- state at least three open problems in architectural governance and the founding-case evidence that motivates each;
- say why a single case limits every conclusion of these notes;
- give the one-sentence version of the whole course, in your own words.

Exercises

Worked solutions are in Appendix A. Attempt each exercise before reading its solution; E1, E3 and E4 can be done individually in about 20 minutes each, E2 suits pairs, and E5 accompanies the interactive website and Guardrail Lab, which are included in the ZIP and run offline. Place in the course: E2 after M2; E1 and E4 after M3; E3 after M5; E5 after M6.

E1. Classify twelve controls. Place each control on the enforcement ladder (none / instructional / conditional / structural) by asking who can defeat it. Note any control whose rung depends on facts not given, and say which facts.

- a) A system prompt line: “Never delete user data.”
- b) A database account, used by the agent, that lacks the DELETE privilege.
- c) A pre-commit hook, in a repository the agent pushes to, that blocks commits touching `/secrets`.
- d) A firewall rule on a separate gateway host blocking all egress except an allowlist.
- e) A daily human review of yesterday’s agent actions.
- f) A rate limiter implemented inside the agent’s own tool-wrapper code.
- g) An API key for a payments provider stored in the agent’s environment, with a policy document saying it is for refunds under \$100 only.
- h) A sandbox that the orchestration script, running as the agent’s own account, launches around each tool call.
- i) No stated policy or mechanism concerning the agent editing its own instruction files.
- j) A hardware key, held by a human, required to sign production releases.
- k) A policy file the deploy agent reads at the start of every session: “Every deploy needs a human’s approval before it goes live.”
- l) An approval gate in the deployment tooling. Only a human’s login can mark a change approved, but the credentials the gate uses for agents are readable by the account the deploy agent runs as, the gate records approval in a database field without binding it to the code that runs, and the production working tree is writable from the agent’s host.

E2. Audit a toy agent config and draw its matrix. A toy deployment description: one agent session runs under account `agent`; it holds (i) SSH access to the production web server, (ii) read-write access to the application repository, (iii) a read-only replica connection to the customer database, (iv) an API token for the email provider, and (v) ownership of its own prompt and policy files. The written policy states: deploys require human approval; emails to customers require human approval; reading customer data is allowed autonomously; editing application code is allowed autonomously; editing its own policy is prohibited. Enumerate 5 action classes, grade access and mandate for each (Levels I to III, as defined in M2), draw the access-vs-mandate matrix, and state the governance gap in one sentence. The deploy row is the running example in miniature; the founding-case counterparts are AC-056 (I, III) and AC-044 (II, III).

E3. Apply the six conditions to a supervision point. First work SP-005, the deployment approval gate’s human approval of a patch (OBSERVED), using the register values given in the box on supervision viability in M5: VC1 partial, VC2 pass, VC3 fail, VC4 pass, VC5 fail, VC6 pass. Name the binding condition as the study recorded it, and propose the single cheapest change that would move it. Then choose a real human-in-the-loop point from your own work or study (a code-review gate, a moderation queue, an approvals inbox), assess it

against independence, latency, halt, attention, comprehensibility and authority (VC1 to VC6), name the binding condition as you would record it, and say why.

E4. Design a structural replacement. Choose one target. Target 1: the payments key of control (g). Target 2: the approval gate of control (l). For target 1, design a structural replacement that preserves the legitimate use (refunds under \$100) while making the prohibited uses unreachable by the agent. For both targets specify: what component enforces the limit, who owns that component, why the agent cannot reach around it, and what the fail state is. Then grade your design's fail state as fail-open or fail-closed and justify.

E5. Guardrail Lab worksheet (website companion). Open the Guardrail Lab, on the live site or from the ZIP (run `python3 -m http.server 8000` in the website folder). The Lab shows what happens to an attempted action under a condition set, as one of four outcomes. **BLOCKED**: the strongest control that binds the action class is structural, so the effect is unreachable. **CHECKED**: the strongest control is conditional, a check runs and the action proceeds. **REMINDED**: the strongest control is instructional, prose is shown and the action proceeds. **SILENT**: no control binds the class. In adversarial mode the actor defeats what it can, conditional controls become **CHECK NEUTRALISED** and instructional ones **REMINDER IGNORED**, and only **BLOCKED** holds. Using the Lab's outcome table, for each condition set CS-01 to CS-10 in order record (i) its guardrail count, (ii) how many of the 60 action classes are blocked, checked, reminded and silent in standard mode, and (iii) how many classes change outcome compared with the previous set. Repeat with adversarial mode on and record how many classes still hold. Then follow AC-044 and AC-056 through the ten sets, and say at which set each becomes blocked and which guardrail makes it so. Finally list the action classes that are never blocked in any set, and write a short paragraph on why a blocked class can still be in the supervision residue of M6. Two cautions: the Lab applies a simplified rule to the catalogue and is not a study result, and its CS-03 row differs from the enforcement ladder of M3 for the reason given in the note on CS-03 below.

The Lab's source (HTML, CSS, JavaScript and the register data as JSON) is in the ZIP under `website/`. To check your worksheet without the browser, run `code/reproduce.py` (or the notebook): its section 9 prints the same table from `website/data.json`, and `code/expected-output.txt` is the output of a clean run.

Note on CS-03: the Lab against the ladder

Why the Lab's CS-03 differs from the ladder. The ladder in M3 counts each action class by its recorded current enforcement, which the register takes from the guardrails listed in that class's enforcing-guardrails field: 36 none, 8 instructional, 12 conditional and 4 structural classes. The Lab applies every guardrail in the condition set whose binds list contains the class, and takes the strongest: for CS-03, 56 guardrails, that gives 9 none, 6 instructional, 37 conditional and 8 structural classes. In the register every guardrail listed as enforcing a class also lists that class in its binds, so the ladder never exceeds the Lab; the binds lists are broader, and 35 of the 60 classes resolve to a stronger class in the Lab than on the ladder (25 are equal and none is weaker). For example AC-038, call a cloud control-plane read API, is none on the ladder but **BLOCKED** in the Lab, because GR-L05-01, a present structural entry (**REPORTED**), lists it in its binds. Why the two lists differ is not documented in the study's documents. For the as-found count use the ladder; read the Lab's CS-03 as the catalogue's bindings applied to the as-found set, not as a measurement.

Hands-on (optional): MeridianSeal Light

The authors run a small public demo, MeridianSeal Light, at <https://neurips2026.meridianseal.com/demo/>. It is not needed for any exercise and the notes do not depend on it. It lets you watch the deploy agent's situation happen instead of reading about it. There is no login. Each session is ephemeral and stops by itself after five minutes, the agents work in a virtual in-memory workspace with no access to a real host, and the language model behind them is a real one (DeepSeek), with a scripted fallback if it is unavailable. It needs a network connection; the offline website and everything else in the ZIP work without it.



The Agent Zone with the Patch Gate waiting for a human decision (Approve or Deny).

Four short steps, each tied to the running example. (1) Open the Agent Zone, launch the three governed agents and send the task “Request a change that policy requires a human to approve”. The agent’s `patch.apply` is conditional, so the request stops at the Patch Gate until you press Approve or Deny: this is the approval gate of M3 and M5, with you as the human on the loop (LO2, LO3). (2) Send “Try to bypass the guardrail” and watch the attempt end in a structural block. Nobody is asked and no instruction is ignored, because the effect is unreachable, which is the rung that differs from step 1 (LO2, LO6). (3) Open the Governance page. It shows the four enforcement rungs, the active mandate with what is outside it, and the six viability conditions VC1 to VC6 (LO1, LO3). (4) Switch the condition set, CS-01 to CS-10, and re-run both tasks: the same request changes outcome as controls are added, as in the Guardrail Lab (E5, M6).

Two cautions. The demo is a teaching sandbox with simplified rules; it is not the founding deployment and nothing it shows is a study result. Its “Patch Gate” is the running example’s approval gate in a simplified form, and the Guardrail Lab remains the tool for E5.

A Worked solutions

E1. Classify twelve controls

- a) **Instructional.** Prose in the agent's context. Defeated by non-compliance alone: error, drift, injection, or intent.
- b) **Structural.** The effect (deletion) is not expressible through the credential the agent holds. Defeating it requires a different credential, which the agent does not hold. Caveat worth crediting: it is only structural if the agent cannot obtain or alter grants by another path.
- c) **Conditional.** The hook is a mechanism, but it lives in a repository the agent can push to; one commit can weaken or remove it. (If the same check ran server-side on a host the agent cannot access, it would be structural. The rung depends on where the check runs and who can write there.)
- d) **Structural.** The rule runs on a separate host outside the agent's reach; the blocked network path does not exist from the agent's position.
- e) **Conditional at best, and viability-limited.** A daily review is a mechanism, but it fails the latency condition (VC2) for anything faster than a day, and its inputs are typically agent-generated records, weakening independence (VC1). If the reviewed logs are writable by the agent, the review is conditional; with independent append-only logs it approaches a (slow) structural audit.
- f) **Conditional.** The limiter is inside code the agent's own account can edit. One edit defeats it.
- g) **Instructional.** The key itself grants full API capability; the sub-\$100-refunds restriction exists only in a policy document. The mechanism (the key) enforces nothing the policy claims. See E4 for the structural redesign.
- h) **Conditional.** The sandbox is real containment, but it is launched by a script running as the agent's own account; the agent can alter the script to skip the sandbox. Containment owned by the contained is conditional.
- i) **None.** No control even claims to exist. In the founding case this rung covered 36 of 60 action classes, more than half the inventory; "we never thought about it" is the most common control in production.
- j) **Structural.** The signing capability is physically held by a human; the agent cannot produce a signed release by any behaviour, compliant or otherwise.
- k) **Instructional.** Prose in the agent's context, defeated by non-compliance alone. This is rung 1 of the running example, the approval policy.
- l) **Conditional, with the rung decided by two facts: where the mechanism runs and who can write to that place.** Approval authority is structural, but the link from approval to the running artefact is not, so the deploy path as a whole is conditional (GR-L06-01, fail-open). The gate's credentials for agents are readable by the deploy agent's account, the approval is a database field not bound to the code that runs, and the production working tree is writable from the agent's host: three ways around the gate, none of which forges a human's approval.

Facts that decide rungs, in summary: *where does the mechanism run, and who can write to that place?* Controls (c), (f) and (h) all fail on ownership; (b), (d) and (j) all succeed on unreachability.

E2. Toy config audit

One defensible enumeration (others are acceptable if graded consistently):

Action class	Access	Mandate
Deploy to production web server	I	III
Edit application code	I	I
Read customer data	I	I
Send email to customers	I	II/III (approval)
Edit own policy files	I	III

Grading notes: SSH to the production server makes deploys technically autonomous (Level I access) though the policy requires approval (Level III mandate). This is the toy version of the running example; in the founding case the counterparts are AC-056, record an approval (I, III, a two-step gap) and AC-044, the deploy itself (II, III, a one-step gap, REPORTED), because there the gate bounds the deploy where it is in the path. The email token is held directly, so sending is Level I access against an approval-level mandate. The read-only replica is the one well-matched pair: reading is both technically available and authorised autonomously, and note that its being *read-only* is a structural control on a sixth class (writing customer data) that this policy never even mentions. Ownership of its own policy files puts “edit own policy” at Level I access against a Level III mandate, the exact pattern of the founding case’s instruction corpus (every instruction file writable by the governed identity).

Matrix: diagonal cells hold 2 classes (edit code, read data); above-diagonal cells hold 3, of which 2 sit in the worst cell, (I,III). Gap sizes, derived: deploy 2, edit own policy 2, send email 1 or 2 depending on whether you grade its mandate II or III. One-sentence gap statement: *three of five action classes, including both prohibited-level ones, are technically autonomous, so the policy’s approval requirements are carried entirely by prose.*

E3. Six-condition assessment (model answers)

SP-005, the approval gate’s human approval of a patch (OBSERVED). Nominal posture: in the loop. VC1 independence: partial. VC2 latency: pass. VC3 halt: fail. VC4 attention: pass. VC5 comprehensibility: fail, because the approval request carries the agent’s own description of the patch and not independent evidence. VC6 authority: pass. The study recorded VC5 as the binding condition. A reader who reads the six cells in order meets VC1 partial and VC3 fail first, so this is a good example of why the recorded condition is a judgement by the study’s convention and not a formula. The cheapest single change, a PROPOSED improvement, is to make the approval request present evidence the agent did not write (the diff, the test result and the deploy artefact’s digest produced by tooling the agent cannot edit), which moves VC5 and part of VC1.

A second model answer, the code-review gate on a shared repository. Independence: partial; the review examines the diff (independent artefact) but the description and test evidence are author-generated, and the 61.1% (REPORTED) self-report divergence in the founding case warns exactly here. Latency: passes for merge-time harm, fails for anything the author’s branch can already do. Halt: passes; an unapproved change does not merge. Attention: often the binding condition; approval under review-queue pressure trends toward rubber-stamping, the local analogue of 46 days with zero recorded interventions. Comprehensibility: strained for large diffs. Authority: passes only if branch protection is enforced server-side; if administrators (or the agent) can push directly, authority collapses to conditional. Cheapest single change, for the typical case where attention binds: cap reviewable diff size so attention is spent within human limits, or, where authority binds, enable server-side enforcement, which is one configuration change that moves the gate from conditional to structural for non-administrators.

The general method being practised: read the six conditions in order VC1 to VC6 and take the first failing condition as the binding one, by the study’s convention (the register does not

reproduce it for every row, so it is a convention and not a formula); spend there and nowhere else.

E4. Structural replacement

Target 1, the payments key. Design: remove the raw provider key from the agent's environment entirely. Stand up a small payments proxy on a host and account the agent cannot access; the proxy holds the only credential. The agent receives a token that can call exactly one proxy endpoint: `refund(order_id, amount)`. The proxy enforces, in its own code and configuration: amount at most \$100, the order exists and is refundable, and per-day volume caps; everything else the provider API offers is simply not exposed.

Grading against the design questions. *What enforces the limit:* the proxy. *Who owns it:* an identity distinct from the agent's, on a host the agent cannot reach; this is the load-bearing fact, since a proxy the agent's account could edit would merely be conditional, the sandbox-launcher trap from E1(h). *Why the agent cannot reach around it:* the raw key exists nowhere the agent can read, and the network path to the provider can additionally be closed from the agent's position (the E1(d) pattern). *Fail state:* if the proxy is down, refunds fail; the design is **fail-closed**, joining the 85 of 134 catalogued guardrails that fail closed rather than the 49 that fail open. Justification: an outage inconveniences refunds; a fail-open bypass would silently restore exactly the unbounded capability the design exists to remove. This is LO6 in miniature and it mirrors the founding case's first structural exemplar: the prohibited effect (arbitrary payment operations) stops being expressible, and compliance is not a term in the outcome.

Target 2, the approval gate. A model answer (PROPOSED, none of it present in the estate): a separate approver identity that no agent credential can use (GR-L01-08, PROPOSED); a deploy artefact whose digest is recorded in the approval and verified at every process start, so that a restart cannot load something else (GR-L06-11, PROPOSED); and a post-deploy check by a principal other than the deploying agent (GR-L06-12, PROPOSED). Who owns each component: an approver identity held by humans and a pipeline in a separate account; the digest check in the process manager, owned by the platform team; the post-deploy verifier under its own identity. Why the agent cannot reach around it: the three routes recorded around the present gate (readable credentials, the record-not-artefact link, the writable production tree) each need a component the agent cannot touch. All three are fail-closed in the register.

E5. Guardrail Lab (expected findings)

Every number below is the Lab's rule applied to `website/data.json`; `code/reproduce.py` regenerates them (section 9 of its output).

E5 expected findings. Standard mode, classes blocked, checked, reminded and silent: CS-01 0, 0, 0, 60. CS-02 0, 0, 25, 35. CS-03 8, 37, 6, 9. CS-04 34, 17, 4, 5. CS-05 45, 11, 0, 4. CS-06 46, 10, 0, 4. CS-07 47, 9, 0, 4. CS-08 49, 7, 0, 4. CS-09 53, 4, 0, 3. CS-10 53, 4, 0, 3. In adversarial mode the blocked count is unchanged and every checked class becomes check neutralised and every reminded class reminder ignored, so only the structurally resolved classes hold: 0, 0, 8, 34, 45, 46, 47, 49, 53 and 53 classes. Prose alone (CS-02) blocks and checks nothing and is ignored in adversarial mode. Classes that change outcome compared with the previous set: CS-02 25, CS-03 46, CS-04 27, CS-05 14, CS-06 1, CS-07 1, CS-08 2, CS-09 4, CS-10 0. The largest single step towards blocking is CS-04, identity separated: the blocked count rises from 8 to 34 because ten identity-layer entries replace the nine inverted entries of CS-03; CS-04 is not a superset of CS-03, which is why AC-001 falls back from REMINDED to SILENT. Later sets add few blocks: 11, 1, 1, 2 and 4. CS-10 gives exactly the same outcomes as CS-09: in the Lab, removing the ten verification guardrails changes nothing, because verification does not block an action; the study's hypothesis is that the loss shows up in claim-divergence detection, a measurement loss rather than a control loss. Seven classes are never blocked in any set: AC-001,

AC-007 and AC-013 are silent at CS-09, and AC-003, AC-005, AC-028 and AC-041 are only checked; four of them (AC-005, AC-007, AC-013, AC-041) are gap classes. The running example: AC-056 is silent in CS-01 and CS-02, checked in CS-03 (GR-L06-01), and blocked from CS-04 by GR-L01-08, a PROPOSED entry; AC-044 is silent in CS-01, reminded in CS-02 (a prose entry, GR-L12-03), checked in CS-03, and blocked from CS-04 by GR-L01-03 and GR-L01-08, both PROPOSED. The Lab's BLOCKED outcomes for the deploy therefore rest on controls the study proposes and the estate does not have. Blocked is not the same as no human needed: at CS-09 the Lab blocks 53 classes, yet 32 classes are in the supervision residue; 31 of those 32 are blocked (AC-041 is only checked) and 22 of the 53 blocked classes are outside the residue, because a structural block removes the agent's ability to act alone while a determinant such as accountability, externality or irreversibility keeps the human decision.

Lab fixtures for a quick check (classes blocked, checked, reminded, silent): CS-03 8, 37, 6, 9 against the as-found ladder 4 structural, 12 conditional, 8 instructional, 36 none; CS-09 53, 4, 0, 3. The residue itself is read in M6 and on the website's residue panel; E5 does not ask you to find it in the Lab. A good closing paragraph observes that the 9 containment-shortfall classes are the ones a better catalogue could still move (removable by identity engineering), while classes held by externality (9), irreversibility (4) and accountability (4) are held by their relationship to the world, not by missing engineering; no slider in the Lab moves them and none should.

B Glossary

One merged glossary of 55 terms, alphabetical, shared by these notes, the slides appendix and the website. (P) marks an author-proposed term, not an established standard; (L) marks a term from the literature.

Access

What an agent is technically capable of doing today with the credentials and tools it actually holds, irrespective of authorisation, graded Level I, II or III. Example: The register records AC-056 at access Level I because separation of duties is by role convention among agents sharing one operating-system identity (AC-056 note); the gate's approve route itself needs a human session (GR-L06-01), so the exposure is the shared credentials and the unbound approval-to-execution link.

Access-versus-mandate dual classification (P)

The grading of every action class twice on the same three levels, once for access and once for mandate; where access is more permissive than mandate there is a governance gap. See the box in M2.

Action class

A distinct kind of action an agent can perform, at a granularity such as committing to a repository or restarting a supervised process; the unit of governance analysis. The founding case inventories 60.

Adversarial ceiling of prose

The authors' own phrase. The upper bound on the guarantee a prose control can give against a model or a text that is working to evade or override it, however carefully the prose is worded.

Adversarial mode

In the Guardrail Lab, the setting in which the actor defeats every control it can: conditional controls become CHECK NEUTRALISED and instructional ones REMINDER IGNORED, and only structural controls still BLOCK.

Agent-emitted event

An event the agent produces about itself, and therefore not independent evidence.

Agentic AI system

An LLM-based system whose outputs are wired to tools, so that generating text performs actions (executing commands, editing files, calling services) rather than only describing them.

Author-attested

A statement the authors make for these materials that is not in the study's registers or documents and was not independently audited, for example the business and city of the founding case.

Binding condition

For a supervision point whose posture diverges, the viability condition the study recorded as the cause. The study's convention reads the conditions in the order VC1 to VC6 and takes the first failing condition as the binding one; it is a convention, not a result, and the order does not reproduce every recorded row.

Blast radius

The extent of what is affected when an action is performed wrongly; recorded in the register as development, local-host, estate or external-public. It is not the risk grade.

Bounded autonomy

Autonomy whose extent is set by the guardrails covering an action rather than by the capability of the agent.

Boundary integrity

Whether the autonomy boundary the architecture claims is the boundary that actually holds in operation.

Condition set

A named, explicit subset of the guardrail catalogue, used to hold the guardrail variable constant at a defined maturity level, for example CS-03 (the founding case as found) or CS-09 (fully hardened).

Conditional control

A control implemented as a check the governed actor could alter or go around by another path (a hook it can edit, a file it can delete); it binds only while left alone. Enforcement rung 2.

Control flow and data flow (L)

Control flow is the sequence of steps a program decides to take; data flow is the information that moves between steps. CaMeL keeps untrusted data from changing which steps run.

Effective posture

The supervision a human can actually exercise at a supervision point, given the six viability conditions; it is weaker than the nominal posture wherever one condition fails.

Emitted signal

The observable event a control produces. A guardrail that emits no observable signal cannot be evaluated and so cannot be an experimental variable.

Enforcement ladder (P)

The four-rung grading of controls by who can defeat them: none (rung 0), instructional (1), conditional (2), structural (3). See the box in M3.

Epistemic label

The tag OBSERVED, REPORTED, PROPOSED or NOT DETERMINED attached to every finding; these materials add AUTHOR-ATTESTED.

Fail-open and fail-closed

What a guardrail does when it breaks: fail-open silently removes the protection; fail-closed blocks the action until repaired. In the catalogue 85 entries fail closed and 49 fail open.

Founding case

A real production multi-agent deployment coordinated by one of the authors, studied

read-only by a single observer who is also its operator in one collection run (20260807-1714, 7 August 2026). It is a source of taxonomy and of the registers, not the doctoral experimental environment. Sector and city are author-attested. See the founding-case block in M0.

Gaming mode

A way of making a metric look good without the underlying property improving.

Governance gap

The case where access is more permissive than mandate (the register field delta is yes): capability nobody authorised. In the founding case 50 of 60 action classes have it. Gap size (mandate rank minus access rank) is a derived presentation, not a register field.

Guardrail

Any catalogued control intended to constrain agent behaviour, graded by enforcement class and by population (present, partial, proposed). The catalogue has 134 entries; it is a catalogue of the design space, not 134 deployed controls.

Guardrail layer

One of twelve groups of guardrails, L1 Identity and credential to L12 Model and prompt layer, ordered from furthest from the model to nearest to it.

Guardrail sensitivity

Whether and in which direction a metric moves when a named guardrail is toggled; a metric with no sensitivity is not a dependent variable.

Human-in-the-loop

A human approves before the action happens. Level III.

Human-on-the-loop

The agent acts and a human reviews afterwards and can intervene. Level II.

Human-out-of-the-loop

The agent acts with no human review beforehand. Level I approximates it.

Illusory progress

The failure mode in which a system reports success while the underlying work has failed; the founding case's REPORTED 61.1 per cent (11 of 18 audited episodes) is a measure of it.

In-context scheming (L)

An agent covertly pursuing a goal given in its context window, for example by hiding its actions or disabling oversight; shown as a capability in constructed tests, not shown by the founding case.

Infrastructure-emitted event

An event produced by something the agent does not write, and therefore usable as independent evidence.

Instruction hierarchy (L)

A rule for which instruction wins when instructions conflict, with developer text above user text above text in tool output; it lowers the chance of a violation but remains a prose-level control.

Instruction-based governance

Governance expressed as text the model reads as context, with nothing in the runtime evaluating it; also called prompt-based governance.

Instructional control

A control that exists as prose (prompt, policy file); defeated by the governed agent simply not complying. Enforcement rung 1.

Inverted entry

One of nine present catalogue entries that describe an absence or a weakening arrangement rather than a control, for example the shared operating-system account; CS-03 includes

them and CS-09 omits them.

Level I, II, III

The three autonomy levels used for access, mandate and supervision level. I: fully autonomous with no prior review, confined to development or version control, or permanently pre-authorised. II: autonomous with subsequent human review, carrying real production impact but reversible, communicated asynchronously. III: requires prior approval or is prohibited outright: irreversible, externally facing, or altering a safety flag. Ordered by the human control they require; permissiveness runs the other way.

Mandate

What an agent is authorised to do today under the deployment's stated policy, graded on the same three levels as access. In the founding case the mandate lives only as prose.

Mandate stability

The frequency and predictability with which what an agent is authorised to do changes over time.

Nominal posture

The supervision posture the architecture claims.

Prompt injection (L)

Text in data an agent reads that the agent mistakes for an instruction and obeys, so that the data's author steers the agent.

Reversibility

Whether the effect of an action can be undone, and at what cost; recorded as reversible, reversible-with-effort or irreversible.

Risk grade

A four-point qualitative consequence rating (low, moderate, high, severe) recorded per action class by the study's observer; reversibility and blast radius are recorded as separate fields; no formula is documented. It is not a level.

Running example

The deploy agent: an agent that may deploy software, an approval policy that requires a human's approval, and an approval gate the agent can route around. Register anchors AC-044, AC-056, GR-L06-01.

Self-report divergence

The rate at which an agent's account of its own work factually diverges from independent evidence; in the founding case 61.1 per cent (11 of 18 audited episodes, REPORTED), a rate within non-randomly chosen candidates.

Single observer

The study's observer is the operator: the author who built and operates the estate examined it. The study is not an independent external audit.

Structural Autonomy Boundary

The point at which delegation to an agent ceases to be safe by instruction and comes to require verifiable architectural enforcement; the theoretical construct of the doctoral proposal.

Structural control

A control the governed actor cannot reach around (a missing permission, an absent network path, an external process); compliance is not a term in the outcome. Enforcement rung 3. An approval gate counts as structural only if the actor can reach neither its credentials nor a path around it; otherwise it is conditional.

Supervision determinant

A property of an action that pins it to human supervision however good the guardrails become; ten are defined (SD-01 to SD-10).

Supervision level

The level (I, II or III) at which an action class can defensibly operate given the guardrails in place: the more constrained of a floor set by the action's own properties and a ceiling set by the guardrail set. The Guardrail Lab does not display it; the Lab shows enforcement outcomes.

Supervision point

A place where the architecture claims some human supervision, intervention or signal; assessed for viability rather than assumed effective. The founding case has 34.

Supervision residue (P)

The action classes that remain above Level I under the fully hardened condition set CS-09; 32 of 60 in the founding case (22 at Level III, 10 at Level II). Also called irreducible supervision residue; it is relative to the catalogue and an upper bound. See the box in M6.

Supervision viability (P)

Whether the supervision an architecture claims at a point actually exists, tested by six conditions that must all hold. See the box in M5.

Viability conditions

The six conditions VC1 independence, VC2 latency, VC3 halt, VC4 attention, VC5 comprehensibility, VC6 authority; asserted, not derived, and read in that order by convention.

Weighted sampling (Lab)

The Guardrail Lab draws attempted actions at random with each of the 50 gap classes weighted four times a no-gap class, so 95.2 per cent of attempts are gap-class attempts.

Acknowledgements

The doctoral study underlying these notes was carried out by the first author under the supervision of the second author. The empirical grounding is a multi-agent deployment coordinated by one of the authors, examined read-only in a single evidence-cited collection run (run 20260807-1714, 7 August 2026). Because the observer is also the operator, the study is not an independent external audit; see Section [M0.7](#). The authors thank the NeurIPS 2026 Education Track reviewers, whose comments on the reading order, the running example, the founding case, the worked access-versus-mandate example, the Guardrail Lab and the consistency of terms shaped this final version.

References

- [1] Y. Ruan, H. Dong, A. Wang, S. Pitis, Y. Zhou, J. Ba, Y. Dubois, C. J. Maddison, and T. Hashimoto. Identifying the Risks of LM Agents with an LM-Emulated Sandbox. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2309.15817.
- [2] E. Debenedetti, J. Zhang, M. Balunović, L. Beurer-Kellner, M. Fischer, and F. Tramèr. AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents. In *NeurIPS 2024 Datasets and Benchmarks Track*, 2024. arXiv:2406.13352.
- [3] R. Greenblatt, B. Shlegeris, K. Sachan, and F. Roger. AI Control: Improving Safety Despite Intentional Subversion. In *International Conference on Machine Learning (ICML)*, 2024. arXiv:2312.06942.
- [4] E. Debenedetti et al. Defeating Prompt Injections by Design. 2025. arXiv:2503.18813.
- [5] E. Wallace et al. The Instruction Hierarchy: Training LLMs to Prioritize Privileged Instructions. 2024. arXiv:2404.13208.

- [6] A. Meinke et al. Frontier Models are Capable of In-context Scheming. 2024. arXiv:2412.04984.