AVM · A WHITE PAPER

The Decision You're Not Allowed to Get Wrong

Why the "AI agents versus business rules" debate has a right answer, and what it means for the software you ship

THE ARGUMENT IN ONE LINE

An AI agent should orchestrate the messy work around a consequential decision. It should almost never make that decision alone. The interesting part is not the rule itself. It is that the same principle governs a layer nobody is watching closely enough: the AI agent that now writes the code your decisions run on.

1. A debate that sounds like a fashion cycle, and isn't

Walk into any enterprise architecture review in 2026 and you will hear some version of the same disagreement. One camp wants to hand a business process to an autonomous agent and let it reason its way to an outcome. The other camp points at the rule engine that has adjudicated that same process, correctly and boringly, for fifteen years, and asks why anyone would replace something that works and can be audited with something that guesses.

The framing that usually gets attached to this is "agents are the future, rules are legacy." That framing is wrong, and being clear about why it is wrong is the most useful thing an architect can do this year.

Rules and agents are not competing to occupy the same spot. They occupy different layers of the same system. The real question was never "which technology wins." It is narrower and much more useful:

> ***Which decisions in my business am I willing to let be non-deterministic?***

Once you ask it that way, the whole thing resolves into an engineering problem instead of a preference. Some decisions are non-deterministic by nature. Is this customer complaint an escalation or a vent? What is the actual intent buried in this vaguely worded request? A rule cannot answer those without a brittle thicket of special cases, and that is exactly the work an agent is good at. Other decisions are the opposite. Does this loan applicant clear the debt-to-income threshold? Does this transaction trip a sanctions screen? Those have a knowable, written-down answer, and the only acceptable behavior is to apply it the same way every single time and be able to show your work afterward.

The mistake is not choosing one side. The mistake is letting the boundary between those two kinds of decisions blur.

2. The asymmetry that should decide almost everything

There is a single property that explains most of the sensible deployment advice in this space, and most teams discover it the expensive way.

Rules fail loudly. When a deterministic rule hits an input it was not built for, the process stops. Someone gets paged. The failure is annoying, visible, and immediately fixable.

Agents fail quietly. When a reasoning model gets something wrong, the process usually completes anyway. It produces a fluent, confident, plausible answer that happens to be incorrect, and it hands that answer downstream as though nothing happened. The failure is invisible until the consequences surface, which may be weeks later, in an audit, or in a headline.

For a low-stakes, reversible decision, quiet failure is a tolerable price for flexibility. For a decision with regulatory exposure, financial consequence, or a discrimination liability attached to it, quiet failure is categorically worse than loud failure. A process that stops is a nuisance. A process that quietly does the wrong thing at scale, while generating records that make it look correct, is the thing regulators exist to punish.

This is not a hypothetical worry. In a controlled study of an LLM-driven pricing agent given a dual mandate to grow revenue while protecting brand equity, the agent, run without hard constraints, overreacted to how its instructions were phrased, cut prices aggressively, and drove a simulated annual loss of roughly one hundred thousand dollars before anyone would have caught it in a real deployment (arXiv 2510.23682, 2025). The agent was not broken. It was doing exactly what a probabilistic reasoner does when you remove the guardrails: producing locally plausible actions that violate the long-term logic of the business.

Hold onto that phrase, “locally plausible, globally wrong.” It is going to describe more than pricing agents before we are done.

3. The synthesis the field has actually converged on

Once you accept the asymmetry, the architecture stops being a matter of taste. The pattern that the decision-management world, IBM’s decisioning group, and a growing body of academic work have all arrived at independently is the same one, and it is worth stating precisely because most popular writing on the topic misses it.

> ***The agent does not make the decision. The agent orchestrates around the decision.***

The division of labor goes like this. The agent handles unstructured input, works out intent, chains across systems, deals with the small fraction of cases that have no clean path, and explains the outcome in plain language afterward. The deterministic layer, rules plus validated scoring models, applies policy to the facts the agent extracted, produces the actual verdict, and generates the auditable record. Agents handle the mess. Rules handle the verdict.

Loan origination is the example the field keeps returning to, because every layer is visible in it. An agent ingests the pay stub, the bank statement, and the borrower’s free-text explanation of an employment gap, and turns all of that into structured fields. It calls the credit bureau, pulls property data, retrieves prior applications. A validated risk model scores probability of default, which is neither a rule nor an agent but a

third thing, a monitored statistical model. Then a rules engine applies the actual policy: debt-to-income limits, fair-lending constraints, product eligibility, jurisdiction. That is where approve or decline happens, deterministically and on the record. When a case fits no rule cleanly, the agent assembles the file, summarizes the anomaly, and routes it to a human underwriter with a recommendation. Finally the agent drafts the adverse-action notice in readable language from the rule engine's structured reason codes.

Notice what just happened. The agent touched every step except the one with legal consequence. That placement is the whole design, not a quirk of this particular example.

The academic work now uses harder language for the same idea. One recent architecture describes a deterministic "Symbolic Governor" that supervises a probabilistic "language-model CPU" through a non-bypassable validation loop that intercepts and checks every state transition before the next step runs (arXiv 2510.13857, 2025). Another embeds decision trees as callable oracles that the agent must consult for any high-precision inference, so that every consequential conclusion traces back to a deterministic rule path (arXiv 2508.05311, 2025). Strip away the terminology and both are saying the thing the decision-management veterans have said for years: let the model reason, but do not let it be the final authority on anything you would have to defend to an auditor.

This is also, quietly, where the whole industry's tooling is heading. IBM, Microsoft, Google, and Salesforce are all converging on the same "control plane" idea, a governance layer that registers agents, routes work between them, and logs the entire chain under one audit policy, rather than trusting any individual agent to police itself (industry reporting on multi-vendor control-plane convergence, May 2026). The consensus answer to "agents or rules" turned out to be "agents inside a deterministic frame," and the argument now is mostly about where you draw the frame.

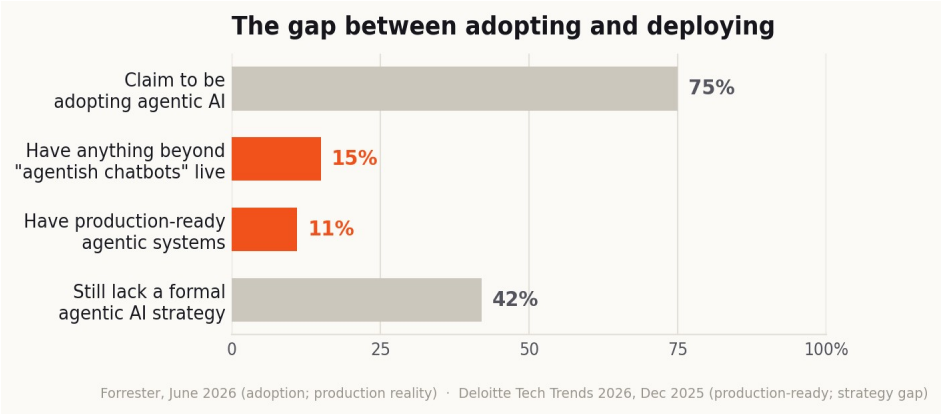
4. The governance gap is measurable, and it is expensive

If this were only an architecture-diagram debate, it would be a pleasant thing for architects to argue about over coffee. It is more urgent than that, because the cost of getting the boundary wrong is now showing up in the numbers.

Gartner's projection has become the most-cited figure in the category: more than 40 percent of agentic AI projects will be canceled by the end of 2027, driven by escalating cost, unclear value, and inadequate risk controls (Gartner, June 2025, restated and widely corroborated through 2026). A year later Gartner sharpened the mechanism, and the sharper version is the one that matters for this argument. By 2027, the firm predicts, 40 percent of enterprises will demote or decommission autonomous agents specifically because of governance gaps discovered only after a production incident. The named root cause is worth quoting: enterprises treat agent governance as binary, either locked down or fully trusted, and that binary is the failure (Gartner, May 2026).

Read that against the argument so far and it lands hard. "Locked down or fully trusted" is precisely the mistake of refusing to separate the layers. An organization that puts an agent fully in charge of a consequential decision has chosen "fully trusted" for something that needed a deterministic frame, and it finds out which decisions those were only when one of them fails in production. The synthesis pattern earns its keep here for a concrete reason: it is the specific control whose absence Gartner is describing.

Deloitte's field data fills in the same outline from the adoption side. As of late 2025, only 11 percent of organizations had production-ready agentic systems, and 42 percent still lacked a formal agentic AI strategy at all (Deloitte, Tech Trends 2026, December 2025). Forrester, in a report pointedly titled "Companies Are Chasing, Few Are Catching," found that roughly three-quarters of enterprise leaders claimed to be adopting agentic AI while only a small minority had anything beyond what the authors called "agentish chatbots" running in real production (Forrester, June 2026). The gap between the ambition and the deployed reality is a governance gap, and everyone measuring it is measuring the same hole.



The orange bars are where the governance gap becomes visible: the distance between what enterprises claim and what they have actually put into production.

5. The layer nobody is governing: the agent that writes the code

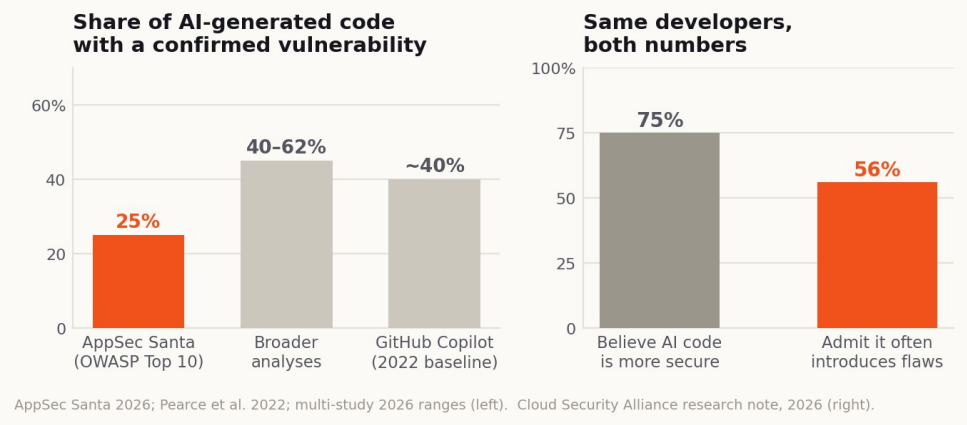
Here is where the argument turns, and where most of the agents-versus-rules conversation stops one step short of the thing that will actually hurt you.

Every version of that debate is about a decision an agent makes at runtime. Should the agent approve the loan, price the product, adjudicate the claim. Fair questions, and the synthesis above answers them. But step back and look at what has changed underneath all of it. An AI agent is now writing the software that implements whichever decision layer you chose. The rules engine, the scoring model, the orchestration glue, the infrastructure it all runs on: increasingly, an agent generated that code.

And almost nobody is governing that agent the way they have learned to govern the runtime one.

The numbers here are not subtle. By 2026, roughly 42 percent of all code is AI-generated or AI-assisted, and developers expect that to pass half by 2027 (Sonar developer survey, 2026). That code carries defects at a measurably higher rate than human-written code. A 2026 study testing 534 samples across six major models against the OWASP Top 10 found a confirmed security vulnerability in one in four of them (AppSec Santa, 2026). Broader analyses put AI-generated code at 1.7 to 1.9 times more likely to carry vulnerabilities than human-written equivalents, with elevated maintainability and logic-error rates that surface after release rather than during testing (Second Talent and CodeRabbit analyses, 2026). The infrastructure layer is worse: 41 percent of AI-generated backend code ships with overly broad permissions, misconfigured IAM roles appear in nearly half of AI-assisted cloud deployments, and AI-generated infrastructure code raises identity-related vulnerabilities by around 28 percent (SQ Magazine security roundup, April 2026).

Now recall the asymmetry from section two, because it applies here with full force. This code fails quietly. It compiles. It passes the tests someone thought to write. It ships. And it carries a vulnerability that traditional review rarely catches, because the reviewer did not write the code and is skimming for plausibility rather than reconstructing intent. There is even a measured human factor: in one study, more than 75 percent of developers believed AI-generated code was more secure than human-written code, while 56 percent admitted it frequently introduced security problems (Cloud Security Alliance research note, 2026). The confidence and the defect rate are both high, in the same population, at the same time. That is the "false sense of security" that security researchers keep documenting, and it is the code-generation twin of the pricing agent that was confidently, fluently wrong.



Left: measured vulnerability rates in AI-generated code across 2026 studies. Right: the confidence-versus-reality paradox, the build-time echo of the runtime problem this paper opens on.

So the debate everyone is having about runtime decisions has a mirror image one layer down, and the mirror image is the one with less attention on it. You can build the most disciplined agents-orchestrate-rules-decide architecture in the industry, and it will still be implemented by an agent that fails silently, in code you cannot easily reproduce, against an environment nobody can fully describe.

> ***Do not let a probabilistic system be the final authority on something you cannot afford to get wrong.***

At runtime, that means a deterministic decision layer. At build time, it means a deterministic frame around code generation. Most of the market has internalized the first and is only beginning to notice the second.

6. Why the obvious fixes don't close it

Before getting to what a deterministic frame around code generation looks like, it is worth being fair to the approaches teams reach for first, because they are reasonable and they each fall short in an instructive way. The strongest version of this argument concedes what actually works before naming what it leaves uncovered.

Bigger context windows and retrieval. The instinct is to give the model more of the codebase so it stops guessing. Retrieval-augmented generation over a code index is the production standard, and it is a sound technique, but its failure modes in large codebases are well documented and structural, not incidental. Context goes stale the moment someone refactors, because the index does not rebuild itself. Files get chunked by token count rather than semantic boundary, so the model sees half a method. And retrieval is

dependency-blind: a hit on a file returns its contents but nothing about what calls it or what breaks when it changes, which in a large system is exactly where the bugs live (Supermemory, June 2026). Even the vendors selling into this space concede that context windows stop scaling somewhere past half a million files, where architectural dependencies overwhelm pattern matching (Augment Code, October 2025). More context helps the model write plausible code. It does not verify the code is correct.

Just set the temperature to zero. The common belief that temperature zero makes a model deterministic is simply false, and this is the single most clarifying piece of research an architect can carry into these conversations. In September 2025, Thinking Machines Lab, the group founded by former OpenAI CTO Mira Murati, published the definitive analysis. The old folk explanation blamed floating-point quirks under concurrent GPU scheduling; that explanation is wrong. The real culprit is batch variance: the numerical reduction operations inside the model produce subtly different results depending on the batch size they happen to run in, and server load makes batch size fluctuate unpredictably. Their measured result is the one to remember: the same prompt run a thousand times at temperature zero produced eighty distinct completions, and only after they rewrote the offending kernels to be batch-invariant did all thousand come back bitwise identical (Thinking Machines Lab, September 2025). Worse for anyone counting on stability, a pinned model name can quietly point to a new snapshot on the provider's side (KeywordsAI, 2025). You do not get reproducibility by turning a knob down. The field's actual conclusion is that you should stop trying to make the model deterministic and instead build a deterministic pipeline around it: generate once, validate, and cache the checked result so downstream systems depend on a verified artifact rather than a fresh roll of the dice (zansara.substack.com, March 2026).

Shift-left policy checks. Mature teams already scan infrastructure code with policy-as-code tools, Open Policy Agent, Sentinel, Checkov, wired into CI to evaluate a Terraform plan before it applies. This is genuinely good practice. But look at where the check sits: it runs after the change has been authored, committed, and pushed, so a violation surfaces in review rather than being prevented at authorship. And the policy-as-code field is honest about its own structural weakness: the enforcement layer is itself infrastructure that can fail, and when it fails, its failure mode is governance-off (Scalr, June 2026). A separate gate bolted alongside code generation is a gate that can be skipped, can fail open, and by construction only ever inspects code that has already been produced.

Each of these is a real improvement and none of them changes the fundamental posture, which is: let the agent produce whatever it produces, then hope to catch the problems afterward. That is prompt-and-pray with progressively better safety nets under it. The nets are worth having. They are not the same thing as refusing to emit bad output in the first place.

7. What a deterministic frame around code generation looks like

The runtime pattern was "agent orchestrates, rules decide." The build-time equivalent is "agent generates, a deterministic pipeline validates and gates." This is the point where it helps to look at a system built specifically around that principle, not as a product tour but because it makes the abstract argument concrete. AVCEL, the governed AI software-delivery system built by the engineering firm AVM, is architected end to end around the idea that a probabilistic generator must sit inside a deterministic frame. Three of its design choices map directly onto the three gaps in the previous section.

It validates against a real replica, not a hopeful approximation. The reason AI-generated code fails quietly is that there is nothing trustworthy to check it against. Staging drifted from production years ago. The documentation was accurate three versions back. The dependency graph describes what was declared, not what actually runs. AVCEL's answer is to stop trusting proxies. A discovery pass the system calls Phenocel reads the existing estate the way a geneticist reads an organism: not only the source code that was written, which the documentation calls the genotype, but the phenotype, the runtime behavior, the real dependencies, the operational knowledge that never made it into a manual. From that it builds a stateless binary replica of the environment, a "carbon copy," and every AI-generated change is executed and validated against that replica before it is allowed anywhere near production. This is a different primitive from retrieval. Retrieval fetches text that looks relevant and lets the model guess; execution against a faithful replica tests whether the change actually works in a copy of the place it will run. The replica is rebuilt from the real estate rather than indexed once and left to rot, which is precisely the answer to the stale-context and dependency-blindness failures that sink retrieval in large codebases.

It makes the pipeline reproducible even though the model is not. AVCEL does the thing the reproducibility research concluded was correct, and it is careful not to overclaim the thing that research proved impossible. It does not pretend the generation step is deterministic; no honest system can. Instead it calls the model once, verifies the output against named paths in the real repository, and caches the verified result under a key derived from the signed-off requirements, the prompt, the model version, and the schema. As long as those inputs are unchanged, every later run returns the same result, byte for byte. Change an input and the key changes with it, so a silent model-snapshot swap invalidates the cache by construction rather than quietly altering output. When an examiner asks you to reproduce a specific AI-generated change from eight months ago, that is an answerable query against the evidence record rather than a promise. The generator is probabilistic. The pipeline is not. That distinction is the whole game, and it is the same distinction the Thinking Machines work points at from the research side.

It validates infrastructure before generation, inside the same pass. Rather than scanning infrastructure code after it exists, AVCEL folds the policy check into the compilation step that produces the change, with a default-deny policy engine sitting in the generation path itself. A violation blocks the build. Ungoverned infrastructure code is not produced and then caught; it is never produced. This closes the specific gap where most AI coding tools validate application logic and leave the infrastructure layer, the overly permissive, IAM-misconfigured layer from section five, entirely unchecked. The gate is not a separate CI step that can fail open. It sits in the path that makes the artifact.

The word AVM uses for all of this is "compiler," and it is worth being precise about the claim because a sharp engineer will otherwise dismiss it. AVCEL is not a compiler in the textbook sense, and its makers do not claim the generation step is deterministic. The compiler analogy is about one specific property: a compiler refuses to emit a promotable artifact unless validation passes. A C compiler does not hand you a binary and hope; it fails at build when the program is ill-formed, where failure is cheap and safe. AVCEL has that same refuse-to-promote property wrapped around a probabilistic core. A prompt fails in production. A compiler fails at build. That is not wordplay. It is a claim about *when* invalid output gets caught, and moving that moment from production back to build is the entire value.

8. Ownership, and the year-four question

There is one more dimension to this that the runtime debate does not surface, and it becomes visible only when you ask what happens after the system is built.

A governed pipeline produces something valuable as a byproduct: the complete, reproducible record of what was built, why, under which policy, and validated against what. That evidence record is the thing your examiners will ask for. It is also, if you are not careful, the thing you can lose.

The market has a category of AI software vendors that deliver finished, governed systems as a managed service. They design, build, host, and maintain the software, which works well until you ask two questions. Where does the finished system run when the work is done? And what does your exit look like if you ever want to leave? With a hosted model, the production system runs in the vendor's environment and the evidence trail lives there too. Your modernized software becomes something you rent access to, and your governance record becomes a dependency on a vendor you cannot easily replace. For a regulated institution, that is concentration risk wearing the costume of convenience.

The deterministic-frame principle has an ownership corollary that is easy to miss. If the whole point is to be able to prove and reproduce your own decisions and your own code, then the proof has to be yours. A governance system you cannot leave is not really a governance system. It is a dependency. AVCEL is built the opposite way from the hosted model: it runs inside the customer's own perimeter, deployable fully air-gapped, and the customer owns the running system, owns the blueprint as a portable and tool-neutral contract, and owns the complete evidence record. What they license is the engine that keeps the blueprint current as the estate evolves. Stop using it and you keep everything already delivered; the only thing you give up is the ability to keep recompiling. Whether or not you adopt a specific product, the principle is the one to carry into any vendor conversation: if the argument for governed AI is that you can prove your own work, then the proof cannot live in someone else's building.

9. The four questions to ask before an agent decides anything

The argument reduces to a short diagnostic. For any decision point, at runtime or at build time, run it against four questions.

Is the logic knowable and written down in advance? If yes, that is rules territory, whether the decision is approving a loan or promoting a code change. If it genuinely depends on judgment over unstructured context, that is where an agent earns its place.

What is the consequence of being wrong? If the answer involves regulatory exposure, financial loss, safety, or discrimination liability, the decision needs a deterministic authority or a hard constraint an agent cannot override. If it is recoverable and low-stakes, an agent alone is fine.

Must you be able to prove why, later, to someone with authority to demand it? A language model's chain of thought is not a legal audit record. If a regulator, auditor, or litigant can compel the rationale, you need a deterministic, versioned, replayable decision path. This is the single strongest argument for keeping the final authority deterministic, and it is non-negotiable in lending, insurance, healthcare, and employment. It is equally non-negotiable for the code that implements any of those.

How often does the logic change, and who changes it? This is where the pro-agent case is strongest and worth conceding cleanly. When rules number in the thousands and change weekly, a system that is auditable in theory becomes unmaintainable in practice, and the flexibility of a reasoning layer is a real answer to real rule-sprawl. The synthesis handles this too: let the agent manage the mutable, messy surface, and keep the deterministic core for the decisions and the code paths you would have to defend.

The through-line is the same at both layers. Let the probabilistic system do the expansive, interpretive, orchestrating work it is genuinely good at. Do not let it be the last word on anything you cannot afford to get wrong, and build the frame that guarantees it never accidentally becomes the last word.

10. Where the debate is still genuinely open

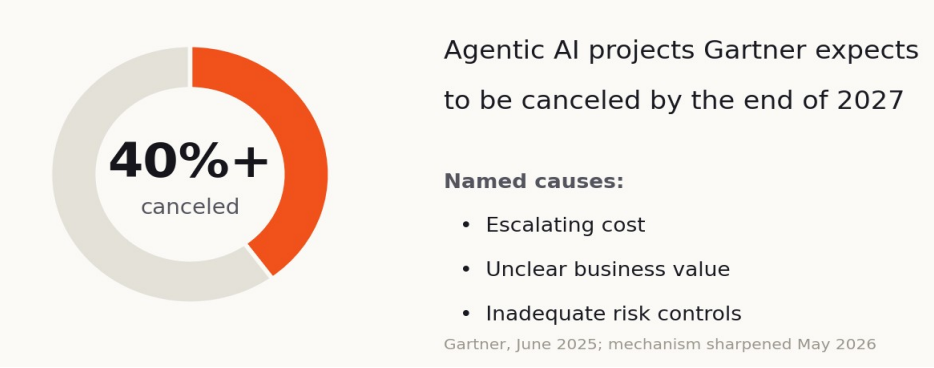
Overstating consensus would undercut the argument, so it is worth naming what remains unsettled, because a reader who has thought hard about this will know these are live.

Does the deterministic layer eventually shrink as models get more reliable and more constrainable? Some argue that everything except explicitly regulated decisions will migrate into the agent over time. That is not resolved, and reasonable people hold both positions. What is not in dispute is that the regulated core is not going anywhere, because the constraint there is legal, not technical.

Is a validated, monitored predictive model a "rule" or an "agent"? Neither. It is a third category, and collapsing it into either side produces sloppy arguments. A risk score is probabilistic like an agent but validated and monitored like a rule, and treating it as its own thing is part of thinking clearly about this.

Does constrained decoding, forcing a model's output into a strict schema, make it "deterministic enough" to trust with more? It narrows the gap and it is the most technically interesting frontier in the space, but a schema constrains the shape of an output, not its correctness. A model can produce perfectly well-formed JSON that is confidently wrong. Structure is not proof.

None of these open questions changes the practical guidance. They sharpen it. The boundary between the probabilistic and the deterministic will move as the technology matures. The discipline of maintaining that boundary, and of never letting a probabilistic system quietly become the final authority on a consequential decision or the code that carries it out, is the durable part. That discipline is what "AI governance" actually means, underneath the word. The organizations that internalize it, at runtime and at build time, are the ones whose agentic projects will still be running in 2028, when 40 percent of the ungoverned ones have been quietly switched off.



Gartner's headline projection, and the reason the discipline in this paper is not optional. The named causes are governance failures, not model failures.

Sources and further reading

Figures are drawn from published research and industry analysis current as of mid-2026. Specific numbers are best verified against the primary source before external citation; several vendor and practitioner sources below are directional rather than peer-reviewed, and are marked where that matters.

Governance and adoption data

Gartner, "Gartner Predicts Over 40% of Agentic AI Projects Will Be Canceled by End of 2027," June 2025.

Gartner, "Applying Uniform Governance Across AI Agents Will Lead to Enterprise AI Agent Failure," May 2026 (the "locked down or fully trusted" binary; decommissioning after production incidents).

Deloitte, Tech Trends 2026, agentic-strategy chapter, December 2025 (11% production-ready; 42% without a formal strategy).

Forrester, "Companies Are Chasing, Few Are Catching," June 2026 (adoption-versus-production gap).

AI-generated code defect and security data

AppSec Santa, 2026 study of 534 samples across six models against the OWASP Top 10 (25.1% carried a confirmed vulnerability).

Sonar developer survey, 2026 (roughly 42% of code AI-generated or AI-assisted).

Second Talent and CodeRabbit analyses, 2026 (1.7–1.9x higher vulnerability and defect rates; post-release surfacing).

SQ Magazine security roundup, April 2026 (41% overly broad permissions in AI-generated backend code; IAM misconfiguration; +28% identity-related vulnerabilities). Directional aggregator.

Cloud Security Alliance research note on AI code-generation vulnerability debt, 2026 (developer overconfidence: 75% believed AI code more secure while 56% saw frequent issues).

Reproducibility and nondeterminism

Thinking Machines Lab, "Defeating Nondeterminism in LLM Inference," September 2025 (batch-variance as the true cause; 1,000 runs at temperature zero produced 80 completions).

KeywordsAI, "Consistent and Reproducible LLM Outputs," 2025 (model-snapshot drift under a pinned name).

zansara.substack.com, "Setting the temperature to zero," March 2026 (validation-and-caching as the correct response). Practitioner source.

Retrieval limits, policy-as-code, and the decision-agent pattern

Supermemory, "Large-Repo Coding Agent Memory Bottleneck," June 2026 (stale context, chunking, dependency blindness).

Augment Code, October 2025 (context-window scaling limits past ~500k files).

Scalr, "Enforcing Policy as Code in Terraform," June 2026 ("enforcement layer can fail; failure mode is governance-off").

arXiv 2510.13857, "A Governance-First Paradigm for Principled Agent Engineering," 2025 (Symbolic Governor over a probabilistic core).

arXiv 2510.23682, "Neuro-Symbolic-Causal Architecture for Robust Multi-Objective AI Agents," 2025 (the unconstrained pricing-agent loss).

arXiv 2508.05311, "Symbolic Reasoning with Decision Trees and LLM Agents," 2025 (decision trees as callable oracles).

Industry reporting on multi-vendor agent control-plane convergence, May 2026 (IBM watsonx Orchestrate and peers).

Avcel architecture

AVM product documentation and avcel.ai/product (Phenocel, the carbon-copy replica, the seven-layer pipeline, reproducibility caching, the model gateway, and the ownership model). Source of record for all Avcel-specific claims above.