Scalable Runtime Governance for Agentic AI in Financial Services

Lukasz Szpruch[1], Agus Sudjianto[2], Tanveer Bhatti[3], and Gary Ang[4]

[1]University of Edinburgh, [2]University of North Carolina at Charlotte, [3]Independent, [4]Quaintitative

April 13, 2026

Abstract. Agentic AI systems, systems that decompose tasks, invoke tools, and execute multi-step workflows, are increasingly deployed in financial services. These systems challenge classical Model Risk Management (MRM) because outcomes arise from *execution trajectories* rather than a stable input-output mapping, and many material failures are *process failures* (unsafe tool use, skipped approvals, privacy breaches, uncontrolled side effects) that emerge during runtime.

We propose an MRM-operational framework that (i) decomposes agentic workflows into reusable *capabilities* (AI components executing a clearly specified bounded set of actions, with explicit authority, constraints, and evidence requirements), (ii) validates and calibrates capabilities using pooled evidence across workflows to improve scalability and consistency, and (iii) extends MRM into *runtime governance* through governance-semantic telemetry, continuous authorisation, temporal policy conformance checking, drift monitoring over trajectory health, and tier-based containment.

Acknowledgements. The authors are grateful to Rafic Fahs (Chief Model Risk Officer (CMRO, FifthThird Bank), Yu Pan (CMRO, US Bank), C.J. Peng (CMRO, Truist), Samir Abrol (CMRO, Sanander US), Ratul Ahmed (Group Head Model Risk Management & Validation, Commerzbank AG), Andrew Harrison (Agentic and GenAI Goverance, ABN AMRO Bank N.V.), Julian Phillips (Global Head of MRM for CIB, HCIB and AI, HSBC), Simon Goo (Head of Group Risk Analytics, Group Risk Management, UOB), Rainer Glaser and Daniel Wang (Oliver Wyman) reviewing earlier versions of this manuscript. Their detailed feedback, critical challenge, and practical insights have significantly strengthened the clarity and relevance of this work.

All views expressed in this paper are those of the authors, and any remaining errors are solely our responsibility.

1 Introduction

Financial institutions are adopting agentic AI to support or automate a wide range of workflows (e.g., KYC review, policy compliance checks, drafting and critique of regulated documents, internal research with attribution). We anticipate agentic AI to become more prominent in financial services over the coming years because it enables goal-directed automation across multi-step operational processes: agents can coordinate retrieval, analysis, drafting, and controlled action execution under constraints, and can be integrated into existing human workflows (approval points, escalation, audit trails) [Aldridge et al., 2025]. Throughout the paper, an *agentic system* is a composition of (i) an LLM, (ii) system prompts or instructions, (iii) a tool layer (APIs, databases, file systems, external services), (iv) memory (e.g., RAG over internal corpora and state

stores), and (v) guardrails (policies and checks), coupled by an orchestration architecture [Khoo et al., 2025]. This architecture expands the operational risk surface: tools and memory introduce side effects and data-access pathways, while orchestration introduces temporal dependencies and emergent behaviours that are not visible from a single input-output snapshot. Indeed the primary object of concern is an *execution trajectory*: a run produces a sequence of intermediate reasoning steps, memory accesses, tool invocations, state transitions, and often human approvals.

The industry's current reliance on prompt-based guardrails or LLM safety filters constitutes a fundamental failure in risk mitigation. These mechanisms present an illusion of control while offering little binding enforcement. Prompt-level guardrails remain advisory. They depend on the model's probabilistic compliance instead of rigorous, enforceable constraints within the execution environment. Even when implemented through additional models or rule-based filters, guardrails are themselves components whose reliability must be tested, calibrated, and monitored. In practice, however, many deployments assume these controls are effective without subjecting them to the same validation and monitoring requirements applied to other model components.

A related and underappreciated limitation is that many guardrail implementations rely on LLM-based verification, implicitly conflating semantic plausibility with logical correctness. These verifiers operate in the same semantic space as the systems they are intended to control: they assess whether an output appears reasonable, well-formed, or consistent with context, rather than whether it satisfies precise structural, numerical, or policy constraints. As a result, fluently written but incorrect outputs can systematically pass such checks. This is not an incidental weakness but an architectural one: when both the system and its verifier rely on probabilistic semantic reasoning, the verifier is often blind to precisely those errors it is intended to detect Sudjianto and Wingyan [2026].

For agentic AI systems in financial services, this distinction is critical. Many decisions hinge on quantities or rules that admit no ambiguity—capital ratios, covenant checks, policy thresholds—where an output is either correct or incorrect, and where plausible but wrong answers can lead to materially different outcomes. In such settings, semantic verification is insufficient by construction. Effective governance therefore requires an explicit separation between semantic and logical verification, with deterministic, rule-based, or formally specified checks applied wherever outputs carry regulatory, financial, or compliance consequences. Crucially, these checks must be embedded in the execution environment and enforced through the system's control structure, rather than delegated to probabilistic model judgement.

From periodic review to runtime governance. Historically, Model Risk Management (MRM) programmes have relied on periodic assessment: models are validated prior to deployment and subsequently monitored through scheduled reviews using aggregate indicators such as performance, stability, and drift computed over batches of production cases [pra, 2023, fed, 2011]. This paradigm becomes increasingly strained for modern AI systems that adapt to changing data and operating environments, and for which risks materialise as sequences of actions over time. Because user inputs, intermediate reasoning, and system interactions are expressed in natural language, the space of possible execution paths is combinatorially large and cannot be exhaustively pre-tested. Minor variations in phrasing or context can lead to materially different execution trajectories [Wicker et al., 2025]. Moreover, even when aggregate performance metrics appear acceptable, severe failures may arise on specific trajectories, particularly where unsafe tool use, missing approvals, or unintended side effects occur during execution [Wang et al., 2025, OWASP Foundation, 2024].

A common response to these risks is the reliance on human-in-the-loop (HITL) review as a

compensating control. However, as agentic systems scale, this assumption becomes increasingly fragile. Human oversight faces three structural limitations. First, agentic systems operate at machine speed while human review operates at human speed, making comprehensive inspection infeasible. Second, reviewers rarely observe the full execution trajectory; instead they see compressed summaries or final outputs, obscuring intermediate reasoning steps, retrieval actions, and tool calls that may carry material risk. Third, complex multi-step workflows generate cognitive overload, making it unrealistic to expect consistent and reliable human verification of every intermediate decision.

These limitations do not eliminate the role of human oversight, but they fundamentally change its function. Rather than acting as a runtime safety net for every decision, human oversight must shift toward policy and workflow design, verification and approval structures, threshold setting, and the review of monitoring evidence. Effective governance therefore, requires mechanisms that operate at the same speed and granularity as the system itself, capable of enforcing policy constraints, verifying actions and outputs, and detecting violations as execution unfolds.

A critical implication of this shift is that human oversight becomes conditional and event-driven. Rather than reviewing every decision ex ante, human intervention is triggered when runtime governance mechanisms detect that human intervention is required, or an execution trajectory violates, or is likely to violate, codified policy constraints associated with the invoked capabilities. These triggers may arise from codified requirements, failed conformance checks, breached thresholds, anomalous trajectory patterns, or uncertainty signals exceeding predefined bounds. In such cases, the workflow transitions into an escalation or containment state (e.g., human approval required, restricted execution, or safe-mode operation). This can be viewed as a form of *perpetual oversight*: the system operates autonomously within governed boundaries, but remains continuously subject to intervention whenever it moves outside its authorised operating envelope. In this way, runtime governance and human oversight are tightly coupled—automation is enabled within policy, while violations dynamically route control back to humans.

We refer to this shift as *runtime governance*: the real-time monitoring and enforcement of policy over an agentic system's execution trajectory. Runtime governance encompasses governance-semantic telemetry, continuous authorisation, temporal and path conformance checking, monitoring of trajectory-level drift, and tiered containment when behaviour moves outside approved bounds.

Reusable capabilities as the basis for scalable governance. Scalable and robust governance requires the right level of abstraction. Banks will deploy a growing number of agentic workflows across business units, often built on shared foundations such as LLMs, retrieval systems, and tool interfaces and either (i) validate each use case independently, which is unscalable and inconsistent, or (ii) attempt to govern individual tools or components, which is brittle under tool substitution (forcing revalidation when implementations change) and fails to capture context-dependent risks arising from how those tools are used within workflows. The central challenge is therefore one of decomposition: agentic AI systems must be broken down into units that are meaningful for governance, yet stable enough to support reuse across use cases.

We argue that *capability* is the appropriate governance abstraction for bank MRM. A capability is a bounded class of actions that an agentic system can execute, defined together with its authority, constraints, and evidence requirements. Capabilities provide this missing contextualisation, enabling reuse across workflows and allowing the same governed abstraction to transfer consistently across multiple use cases. This is motivated by the fact that the risk arises not from the model itself, but from the specific actions the system is authorised to perform.

Indeed, the models without context about how they are used fall outside the scope of meaningful governance.

An agentic AI system can be viewed as a cluster of capabilities serving multiple use cases. Individual workflows orchestrate these capabilities in different sequences, but the underlying capability catalogue remains comparatively stable. This stability allows testing, validation, monitoring, and controls to be defined at the capability level and reused across many applications, while still preserving the contextual information required for effective MRM.

We anticipate a rapidly growing number of AI use cases across financial services, the set of distinct capabilities that must be orchestrated to deliver those use cases is likely to grow much smaller. When new use cases are assembled largely from capabilities that are already understood, tested, and equipped with scalable controls, the incremental MRM work required to sign off the new use case can be streamlined: governance focuses on (i) verifying the use case's capability composition and authority scopes, (ii) checking that the required control tier and approvals are in place, and (iii) confirming that the monitoring and escalation plan is appropriate for the deployment context.

Capability-centric governance also makes interaction between the first and second lines of defence more structured. The 1LoD retains responsibility for the specific use case—its design choices, controls, monitoring and operational outcomes—but, at inception, can communicate to 2LoD which capabilities from the approved catalogue are being invoked, under what authority, and in what operating context. The 2LoD, holding the technical expertise on how each capability is validated, controlled, and monitored across the bank, can then provide the relevant evidence packs, playbooks, known limitations, and current performance/incident signals for those capabilities based on their use in other deployments. This gives 1LoD a clearer picture from the outset of expected performance, the cost and effort involved in controls and monitoring, and the testing that will be required for sign-off. Even where use-case-specific or domain-specific contextualisation is needed (e.g., tightened thresholds for credit-facing applications, domain-specific policy semantics for compliance), the bulk of the validation, testing, and other control infrastructure is already in place from the capability catalogue, and the incremental work is configuration, rather than construction of new control infrastructure.

At the same time, capabilities support failure-mode localisation and learning. Runtime governance is ultimately concerned with reliability across multiple use cases and operating contexts. When a failure mode arises—or when the system stops performing as intended and expected—it is of paramount importance to isolate the root cause (e.g., a specific capability, a particular interaction between capabilities, a tooling boundary, or a telemetry/control gap). Without root-cause localisation, organisations can only apply broad mitigations (tighten everything, add blanket approvals, reduce autonomy), which is costly and often ineffective. In contrast, capability-level decomposition enables targeted remediation and learning: failures can be attributed to specific capability failure modes and to the trajectories that produced them, enabling changes to tests, thresholds, controls, or orchestration that directly reduce the probability of similar failures in the future. Because these improvements are made at the capability level, the benefits extend across all use cases that rely on that capability, not just the individual workflow in which the failure was observed. The formal capability definition and the associated evidence and telemetry requirements are given in Section 2.

Contributions. We introduce a three-layer governance model for agentic AI in banking: the *system level*, where related use cases may be clustered for governance purposes but risk remains owned at the level of each implemented use case; the *capability level*, which provides the

abstraction needed for scalable reuse; and the *trajectory level*, which provides the object of runtime compliance. Specifically we translate the capability layer into a bank-ready MRM programme design, including pooled validation, calibration, evidence standards, and governed change triggers. We translate the trajectory layer into a runtime governance architecture based on governance-semantic telemetry, continuous authorisation, temporal policy conformance, drift monitoring, and tiered containment. Finally, we provide a worked example across three representative use cases and a practical playbook covering artefacts, acceptance gates, monitoring, escalation, and ownership.

Paper structure. Section 2 develops the operational MRM playbook for agentic AI. It introduces the key governed objects—the capability catalogue, use-case specification, execution trajectory, and governance-semantic telemetry—and shows how these support capability-based reuse, trajectory-level policy enforcement, runtime monitoring, risk-tiered control assignment, and explicit change control across the lifecycle. Section 3.2 then illustrates the framework in practice. It first presents a detailed credit memo drafting example to show how runtime governance can be implemented through guarded state transitions, and then generalises to three representative use cases to show how seemingly distinct workflows can be realised as compositions over a shared capability cluster with reusable validation, controls, and monitoring. Section 4 reviews the related literature and positions the contribution. Section 5 discusses limitations and open problems.

2 MRM playbook: capability catalogue, telemetry, and runtime governance

This section develops a playbook to operationalise the key considerations set out in the introduction. The objective is to move from those conceptual requirements to the concrete governance artefacts, enforcement logic, and monitoring substrate needed to manage agentic AI risk in practice. Two implementation requirements drive the design. First, evidence must be *reusable* across a growing portfolio of use cases assembled from shared capabilities. Second, evidence must be *actionable*: it must connect directly to runtime controls that constrain behaviour and produce auditable records.

To make this concrete, we introduce the minimal governed objects required for implementation. These are: (i) the *capability catalogue*, which defines the reusable governance units; (ii) the *use-case specification*, which records how a particular workflow composes capabilities, authority scopes, approvals, and escalation paths; (iii) the *execution trajectory*, which is the object over which runtime compliance is assessed; and (iv) *governance-semantic telemetry*, which provides the evidence needed for control enforcement, effective challenge, monitoring, and incident reconstruction. The remainder of the section shows how these objects are created, connected, and governed across the lifecycle.

A useful way to read the playbook is as a transformation from governance intent into executable control. Human policy is first translated into structured capability specifications and orchestration rules; these rules then govern runtime execution; telemetry emitted during execution becomes the evidence base for monitoring, challenge, escalation, and change control. The layered view clarifies how human policy becomes operational governance for agentic systems.

Layer 1: Human policy. Governance begins with natural-language requirements: regulatory obligations, internal standards, approval requirements, operational procedures, and role-based restrictions. These define what actions may be taken, under what authority, with what approvals, and subject to what constraints on data access, communication, and external side effects.

Layer 2: Capability and orchestration specifications. Policy becomes operational through

two linked artefacts. The first is the *capability specification*, defined as follows.

Definition 1 (Capability)**.** *A* capability *is an AI component executing a clearly specified bounded set of actions. A capability is defined by: (i) the permitted tools/resources and their allowed scope (authority), (ii) constraints that must be enforced (e.g., authorisation, required approvals, prohibited sequences, time windows), and (iii) the evidence required to validate and monitor it (metrics, thresholds, governance-semantic telemetry). A capability may be implemented by one or more nodes in an orchestration graph.*

The second is the *orchestration specification*, represented by the transition system W, which defines which capability invocations are admissible from which governed states, including retries, interrupts, escalation paths, and approval dependencies. Together, these artefacts turn policy into an executable control structure.

Layer 3: Governed execution. At runtime, the system executes a use case as an orchestration of capabilities under the constraints defined in Layer 2. Each run produces an execution trajectory consisting of capability invocations, state transitions, tool calls, approval events, and control decisions.

Layer 4: Telemetry, monitoring, and runtime governance. Execution is instrumented through governance-semantic telemetry. This telemetry is the operational evidence substrate: it allows the bank to enforce policy over trajectories, reconstruct what happened in a particular run, monitor behaviour at the population level, and determine whether thresholds, controls, or approvals remain appropriate as the system evolves.

The sequence of the section follows this operational logic. We first define the governed objects and their notation. We then show how use cases are represented as trajectories over a capability transition system, how telemetry is mapped into production observability infrastructure, and how these foundations support capability cataloguing, onboarding, runtime enforcement, monitoring, and governed change.

Use cases as trajectories over a capability transition system We represent each use case as an orchestration of *capabilities*. This defines a *capability transition system*: states capture a governed abstraction of context (e.g., role, data classification, tier, and whether required approvals have been obtained), and transitions correspond to capability invocations, annotated where needed by tool intent (read/retrieve, compute, write/dispatch) to reflect materially different authority. As a result, runtime control is applied to the unfolding execution trajectory, including the approvals, authorisation decisions, and tool intents associated with each step.

A single run of the use case then produces an execution trajectory (trace)

$$\tau = \big(s_0, a_0, s_1, a_1, \ldots, s_{T-1}, a_{T-1}, s_T)\big),$$

where each element records the governed state before the step, the capability action invoked, and the resulting governed state after the step.

Formally, we represent a use case as an orchestration policy over a finite set of capability calls, written as a labelled transition system $W = (s_0, S, \Pi, \delta,)$ where:

- s_0 is the initial governed context at the start of a run.
- S is the set of *governed context states*—the minimal information needed to make control decisions during execution (e.g., role, data classification, tier, approval status);

- Π is the set of *available next actions* $a \in C$—labels for capability C invocations (optionally annotated with tool intent such as read/compute/write/dispatch to reflect materially different authority);
- $\delta : S \times \Pi \to S$ captures the *orchestration rules*—given the current governed context and a chosen capability step, it determines the next governed context.

A useful interpretation is that reasoning and authority are separated by construction. The model may execute bounded reasoning within a capability, but it does not determine its own permissions, expand its authority, or select arbitrary next actions outside the governed transition system. Instead, policy encoded through capability constraints and orchestration rules determines which actions are admissible and under what conditions.

Crucially, each transition event (s_t, a_t, s_{t+1}) is accompanied by *governance-semantic event metadata* (capability ID/version, tool scope, guard scores/thresholds, authorisation decisions, approval events, and containment actions). This metadata turns the trajectory into an auditable governance object: it links the observed sequence of actions to the evidence needed for monitoring, effective challenge, and incident reconstruction.

Governance requirements are therefore expressed as constraints over trajectories (e.g., "approval must occur before any write/dispatch action", "no external release without sign-off", "no write after sensitive-data flag"). Runtime governance enforces these constraints by monitoring the trajectory as it unfolds and blocking any next step that would violate policy.

The following formalization makes this runtime enforcement explicit. At each step t, the system is in a state $s_t \in S$, as defined by the workflow. Governance decisions operate on measurable attributes of the state. For example:

$$\begin{aligned}
\text{Alignment}(s_t) &\in [0,1] && \text{alignment between the current task and verified knowledge,}\\
\text{Verified}(s_t) &\in \{0,1\} && \text{whether the current result has been independently verified,}\\
\text{Length}(s_t) &\in \mathbb{N} && \text{length of the execution trajectory so far,}\\
\text{Information}(s_t) &\in \{0,1\} && \text{whether new, unverified information has been introduced,}\\
\text{Confidence}(s_t) &\in [0,1] && \text{strength of verification evidence supporting the current result.}
\end{aligned}$$

The state attributes are compared against a set of predefined parameters

$$\Theta = (\theta_{\text{Alignment}},\ L_{\max},\ \theta_{\text{Confidence}},\ \theta_{\text{Escalation}})\,.$$

Here, $\theta_{\text{Alignment}}$ is the minimum alignment required to proceed, $L_{\max}$ is the maximum permitted trajectory length, $\theta_{\text{Confidence}}$ is the minimum confidence required to accept a result, and $\theta_{\text{Escalation}}$ is the confidence threshold below which escalation is triggered. The Boolean attributes $\text{Verified}(s_t)$ and $\text{Information}(s_t)$ are used directly in guard conditions rather than being compared to thresholds.

Definition 2 (Governance Decision)**.** *Every governance decision can be expressed as a deterministic function*

$$f : S \times \Theta \to \{0,1\},$$

mapping the current state and parameters to a binary outcome.

At each step, the system selects an action from a finite set of available actions associated with the capabilities. The selection is governed by a deterministic policy $a(s_t, \Theta)$, which evaluates a

sequence of guard conditions over the current state. Once the action is executed, the system transitions to a new state according to the transition operator δ, $s_{t+1} = \delta(s_t, a_t)$.

At each step, guards are evaluated in a fixed priority order. If the maximum trajectory length has been reached, execution halts. If alignment with verified knowledge is below the required threshold, the system retrieves additional information. If the current result has not yet been independently verified, the system invokes verification. If verification has been completed but confidence remains below the escalation threshold, the system escalates. If the result is verified and the trajectory length remains within the allowed limit, execution may continue. If newly introduced information has been verified, it may be stored. If none of these conditions applies, execution either halts or transitions to a designated fallback state.

Every step records a trace event (s_t, a_t, s_{t+1}), so that each decision can be reconstructed from the state and parameter values at the time it was made. This ensures that governance is not only specified but executed as a deterministic control process over the transition system. In particular, no action can occur unless it is explicitly authorized by the policy, and all transitions are mediated by computable guards derived from the governed state.

Telemetry sufficiency Runtime governance depends on telemetry that is sufficient for (i) automated control enforcement, (ii) effective challenge by 2LoD, and (iii) incident reconstruction. Telemetry is therefore the runtime evidence substrate that makes governable workflows operationally auditable. One run (trajectory) τ corresponds to one *Telemetry trace* [OpenTelemetry, 2026]. Each capability invocation corresponds to one or more *spans* within the trace. Tool calls are spans (or nested spans) with attributes encoding tool intent (read/write/dispatch) and resource scope. Guard evaluations, authorisation decisions, approval events, and containment actions appear as span events and attributes.

This provides a direct implementation path from the abstract notion of "execution trajectory" to a monitoring and control substrate used in production systems.

Practical considerations

In practice, reliability for agentic systems cannot be reduced to a single aggregate success metric. It must be monitored across distinct operational dimensions: consistency, robustness, uncertainty awareness, and safety [Rabanser et al., 2026]. Consistency concerns stability of outputs and decisions under small perturbations; robustness concerns performance under distributional shift, noisy inputs, or adversarial content; uncertainty awareness refers to the ability to quantify uncertainty and anticipate when the system will abstain, escalate, or fail; and safety concerns policy compliance, containment, and the absence of harmful side effects. These dimensions should be monitored both at the level of individual capabilities and at the level of execution trajectories. At the capability level, this includes pass/fail rates, guard score distributions, escalation frequencies, and structured error taxonomies. At the trajectory level, this includes trace length distributions, loop and retry behaviour, unexpected action sequences, policy near-misses, and containment events. Monitoring must therefore operate simultaneously as a reusable capability-level signal and as a context-sensitive use-case signal, with thresholds tightened for higher-risk tiers.

A related practical consideration is how to group capabilities into AI systems in a way that supports reuse, auditability, and root-cause localisation. By clustering we mean grouping capabilities into coherent subsystems that share similar action types, authority scopes, failure modes, control patterns, and telemetry structures. Effective grouping reduces the number of distinct governed objects without losing the link between behaviour, risk, and control. In practice,

this involves grouping capabilities that act on similar resources (e.g., retrieval versus write or internal versus external side effects), exhibit similar dominant failure modes (such as citation errors, numerical inconsistencies, or policy violations), rely on similar control patterns (for example, deterministic checks, allowlists, or approval gates), and produce comparable telemetry signals.

Even when no individual trajectory violates policy, the trajectory distribution may drift over time. For example, an agent may begin to favour shorter paths, rely more heavily on one retrieval source, exhibit different branching behaviour, or abstain less frequently. We refer to this as *orchestration drift*. It should be monitored separately from traditional data drift and model drift, because it concerns changes in execution behaviour over the capability transition system rather than changes solely in inputs or base-model performance.

A critical requirement for operationalising the framework is that governance decisions must be expressible as deterministic functions over the governed state and measurable signals, executing in bounded time and independent of the language model. This requirement defines the boundary between enforceable control and advisory guidance: only controls that can be evaluated deterministically over telemetry can reliably constrain behaviour at runtime. Any control that cannot be reduced to this form should be treated as non-binding and its residual risk explicitly accounted for.

This requirement has direct implications for how verification is designed. In particular, commonly used approaches such as embedding similarity or LLM-based judges are not suitable as primary control mechanisms in high-consequence settings. These methods are structurally misaligned with the types of claims that arise in financial workflows: they cannot reliably represent asymmetric logical relations (e.g., entailment or contradiction), do not enforce numeric constraints, and tend to collapse distinctions such as negation. As a result, they may provide useful signals, but cannot serve as the basis for binding governance decisions.

Verification must therefore be decomposed according to the structure of the claim being evaluated. Capability evidence packs should specify, for each class of output, the corresponding verification mechanism and its correctness guarantee. In practice, this requires:

- relational and logical claims to be verified through structured reasoning over governed representations that support implication, negation, and multi-step inference;
- numeric constraints to be enforced through deterministic inequality checks with defined tolerances;
- exact numerical values to be retrieved from integrity-verified registers with deterministic access and verification;
- logical consistency to be monitored at the trajectory level through explicit contradiction signals.

2.1 Risk tiering, control assignment, and incremental governance

A central requirement for scalable MRM is reducing the marginal cost of onboarding new AI use cases. The practical goal is that new workflow assembled from previously approved building blocks does not trigger a full rebuilding of validation, testing, and controls from first principles. This is achieved by combining (i) a capability catalogue with validated evidence and controls, and (ii) a tiered risk triage that determines the intensity of governance and the required control uplift.

Risk tiering as a function of capability composition and authority. For a use case U, the risk tier is determined by five dimensions:

- **Agency:** degree of autonomy in multi-step execution and adaptation,
- **Authority:** permitted actions (read, compute, update internal records, release externally, commit a business action),
- **Impact:** consequence of failure (financial, regulatory, customer),
- **Exposure:** frequency and openness of interaction with the environment, including reliance on external inputs, user interaction, and deployment scale,
- **Recoverability:** ability to detect, contain, and reverse outcomes.

These dimensions are evaluated jointly, with authority and impact as the primary drivers, and agency, exposure, and recoverability acting as amplifiers. This maps naturally to standard banking tiering terminology while making it operational for agentic systems. Impact corresponds to familiar notions of financial, regulatory, operational, and reputational harm. Authority sharpens this by specifying the actions through which harm can occur, such as writing, dispatching, or transacting. Complexity is captured through agency, namely the extent of multi-step reasoning, planning, and adaptation.

Exposure captures conditions under which failures can occur, including the openness of inputs, susceptibility to adversarial manipulation, and reliance on external data or systems. Importantly, exposure is a property of the use case rather than the control environment: runtime governance mechanisms such as gating, policy enforcement, and containment act to reduce the effective risk associated with this exposure, but do not eliminate it.

Recoverability makes explicit an often implicit consideration in existing frameworks: how easily failures can be detected, contained, and reversed. Taken together, these dimensions reflect familiar regulatory concerns of materiality, reliance, and control effectiveness, but make them directly measurable at the level of workflows.

The key point is that risk depends not only on how a model behaves, but on what the workflow is permitted to do, how exposed it is to failure, and how difficult it is to regain control when things go wrong.

2.1.1 Risk Tiering and Controls

Next we define four operational risk tiers that map directly to governance intensity. Controls should be understood as progressive substitutions rather than literal inheritance: higher tiers preserve the governance objectives of lower tiers—bounded authority, oversight, traceability, and containment—but implement them through stronger and more automated mechanisms appropriate to higher autonomy and consequence.

- **Tier 1: Assistive (low agency, no side effects).** Systems provide recommendations or drafts with no direct execution authority. Human users retain full control over outcomes.

  Typical use cases: summarisation, internal search, drafting support.

  Controls: Data access should be restricted to read-only mode, with the system prevented from taking external actions or creating side effects. Outputs should be evaluated for

issues such as factual errors, information leakage, and potential misuse before they are relied upon. Consequential steps should remain subject to human review before action is taken. Prompts, outputs, and source material should be logged to ensure transparency, traceability, and effective oversight.

- **Tier 2: Bounded workflow (moderate agency, constrained authority).** Systems execute multi-step workflows within predefined boundaries, with limited tool use and reversible actions.

  Typical use cases: internal automation, structured document processing, controlled decision support.

  Controls: The control objective remains bounded authority, traceability, and review, but the mechanism moves beyond simple read-only access and case-by-case human review. Tool usage should be restricted through explicit allowlists and enforced under a least-privilege model, so that only approved tools can be accessed and only with the minimum permissions required. Actions that write, dispatch, or otherwise create external effects should pass through approval gates or equivalent workflow controls before execution. Capabilities should be subject to dedicated testing, with thresholds calibrated to ensure reliable performance under expected conditions. Runtime telemetry should capture tool usage and state transitions, extending basic logging into structured observability. This should be complemented by adversarial testing to identify vulnerabilities and failure modes under stressed or manipulated conditions.

- **Tier 3: High-impact governed execution (high authority or consequence).** Systems influence or execute actions with material financial, regulatory, or customer impact.

  Typical use cases: credit decision support, compliance workflows, external communications.

  Controls: The control objective remains constrained authority, effective oversight, and bounded risk, but governance must no longer rely primarily on human review at the point of action. Instead, execution should be governed through formal risk assessment with explicit likelihood and severity scoring, aligned with risk appetite. Policy-as-code constraints should define and enforce admissible execution trajectories, so that compliance is embedded in the execution environment. Authorisation should be continuous, with tool use and intent checked at each step of execution. Monitoring should operate at both the capability and trajectory level, including near-misses, anomalous sequences, and other signs of control weakness. Drift detection over execution patterns should be introduced to identify changes in orchestration behaviour over time. Strong auditability should be maintained through trace-level evidence and replayability, enabling effective review, challenge, and incident investigation.

- **Tier 4: Critical autonomous systems (high autonomy and irreversible impact).** Systems can independently commit actions with significant and potentially irreversible consequences.

  Typical use cases: autonomous transactions, external commitments, infrastructure control.

  Controls: The control objective remains the same, namely to keep authority bounded, actions observable, and failures containable, but the implementation must assume minimal opportunity for timely human intervention. Authority should therefore be granted on a

default-deny basis, so the system can do nothing unless explicitly permitted, and even then only within tightly bounded scopes. Critical actions should require enforced separation between approval and execution functions across independent roles or control points, and where appropriate, independent dual approval. Execution should be enforced in real time against permitted trajectories, with fail-closed behaviour such that any violation results in immediate interruption rather than continuation. Execution environments should be isolated, with tightly restricted access to tools, data, and external systems. Rollback and containment mechanisms should be mandatory, enabling rapid limitation or reversal of harmful actions. These controls should be supported by high-frequency monitoring and regular governance review, reflecting the elevated consequence and reduced recoverability of such systems.

Incremental residual risk: what changes relative to the approved tiering profile. A new use case should inherit, by default, the validated failure modes, test suites, thresholds, monitoring signals, telemetry schema, and control playbooks of the capabilities it reuses. The incremental MRM task is therefore to identify whether the new use case changes the approved risk profile along the same five dimensions used for tiering: agency, authority, impact, exposure, and recoverability. We refer to this change as the *incremental residual risk*.

For a candidate use case U, incremental residual risk is present only if at least one of the following holds relative to previously approved deployments of the same capabilities:

1. **Agency increase:** the workflow introduces more autonomy, adaptation, or multi-step dependence than previously validated;
2. **Authority increase:** the workflow moves to stronger permissions, e.g. from read/compute to write, release externally, or commit a business action;
3. **Impact increase:** failure would create greater financial, regulatory, customer, or reputational harm;
4. **Exposure increase:** the workflow is used more frequently, at greater scale, with more open inputs, or with greater reliance on external users, data, or systems;
5. **Recoverability decrease:** failures become harder to detect, contain, reverse, or investigate.

Onboarding and sign-off should then answer four approval questions:

1. **Capability and orchestration fit:** Is this use case covered by the existing validated capability catalogue and previously approved orchestration patterns? If not, inherited approval stops here: any new capability or materially new orchestration pattern requires full evaluation.
2. **Tier and control requirement:** Given its profile across agency, authority, impact, exposure, and recoverability, what risk tier should be assigned, and what level of controls, approvals, and monitoring does that tier require?
3. **Incremental control uplift:** Relative to previously approved use cases, do changes in capability composition, agency, or authority require stronger gates, tighter thresholds, or additional approval points?

4. **Runtime governance deployability:** Is telemetry sufficient to support effective runtime governance in production, including policy enforcement, monitoring, escalation, and incident reconstruction?

The governance rule is: *inherit by default, re-evaluate where the use case falls outside the validated capability/orchestration envelope, and uplift controls where the incremental residual risk changes the required tier.*

2.2 Step-by-step programme with explicit 1LoD/2LoD responsibilities

We now revise the programme steps to make responsibilities explicit and to align with the introduction: 1LoD owns the use case; 2LoD owns the capability catalogue, evidence standards, thresholds, and control playbooks; Security/Ops owns tool authority and telemetry integrity.

Step 1: Establish the capability catalogue (2LoD lead; 1LoD input; Security/Ops co-own authority). 2LoD defines the capability taxonomy, the required fields in capability specifications, minimum evidence standards, and approval tiers. Security/Ops defines tool authority models, least-privilege defaults, and logging and retention requirements. 1LoD proposes new capabilities required for business use cases and specifies their intended use and context. The output is a versioned capability catalogue with stable capability identifiers, authority scopes, constraints, and telemetry schema requirements.

The capability catalogue functions as the governed inventory for agentic systems. Each entry serves both as an operational governance artefact and a compliance artefact: it specifies what the capability does, what authority it has, what constraints apply, what evidence supports it, and where it is deployed. This provides the foundation for reuse, effective challenge, and scalable oversight.

Step 2: Create capability evidence packs and control playbooks (2LoD owns standards; 1LoD builds; 2LoD challenges). For each capability, failure modes are identified and linked to likelihood and severity considerations consistent with risk appetite. This is translated into test suites, metrics, thresholds, monitoring signals, escalation rules, and change triggers. 1LoD implements test harnesses, runs evaluations, proposes operating thresholds, and implements controls. 2LoD independently reviews methods and coverage, challenges thresholds, and approves operating points and residual risk.

Failure-mode-driven evaluation becomes a compounding asset at this stage. Incidents and near-misses observed in development or production are mapped back to capability failure modes and promoted into regression tests and monitoring signals across all relevant use cases [Wicker et al., 2025].

Step 3: Use-case onboarding as capability composition (1LoD lead; 2LoD reviews composition and tiering). A new use case is specified as a composition of capabilities, together with associated authority scopes (data sources, tools, write permissions), a control tier determined by likelihood and severity, approval requirements, and a monitoring and escalation plan. Dependencies on third-party systems are also recorded where relevant.

This step defines the governed object and its control configuration: what the system is allowed to do, under what authority, and with what level of risk. It also determines the required controls, but does not yet establish that they are correctly implemented or enforceable.

1LoD owns the use-case design and its outcomes. 2LoD verifies that the selected capabilities, authority scopes, control tier, and approval logic are consistent with policy, and that the required capability evidence packs exist and are applicable.

Step 4: Integration and trajectory conformance (1LoD builds; 2LoD challenges; Security/Ops validates enforcement points). 1LoD implements integration tests for capability composition, encodes temporal and path constraints as policy monitors, and tests trajectory conformance under both representative and adversarial scenarios. The objective is to demonstrate that the governed transition system correctly enforces the intended authority model and prevents prohibited transitions under test conditions. 2LoD challenges coverage, ensuring that policy monitors capture high-severity failure modes and that no material transition patterns are left ungoverned. Security/Ops verifies that enforcement points are correctly implemented, including complete mediation of side effects, fail-closed behaviour where required, and the integrity of telemetry.

This step establishes that governance is not only specified but *testably enforceable* prior to deployment.

Step 5: Deploy with runtime governance (shared). Deployment places the workflow under continuous runtime governance, where the controls validated in Step 4 are actively enforced over execution trajectories. This includes continuous authorisation, tool-intent gating, and real-time enforcement of policy monitors that block prohibited transitions. Tiered containment mechanisms, such as safe modes, tool restrictions, or human takeover, are activated when execution approaches policy boundaries or violates constraints.

In operational terms, deployment means that the system runs under the authority model, transition constraints, and telemetry requirements established earlier in the programme, with all material actions mediated and auditable in real time. Controls are therefore not only validated ex ante, but enforced throughout execution with trace-level evidence.

Step 6: Monitoring and reliability management (1LoD operates; 2LoD oversees; Security/Ops supports). Monitoring is aligned to operational reliability dimensions: consistency of behaviour under small perturbations, robustness under distributional shift and adversarial inputs, calibration of uncertainty signals and escalation behaviour, and safety with respect to policy violations, privacy, and side effects [Rabanser et al., 2026]. In this framework, these dimensions are evaluated through observable properties of the governed state and execution trajectories.

At runtime, monitoring operates over both capability-level and trajectory-level signals derived from the state. Capability-level monitoring tracks the behaviour of individual capabilities through distributions of state attributes and control outcomes, including verification rates (e.g., frequency of $\text{Verified}(s_t) = 1$), distributions of $\text{Confidence}(s_t)$ relative to acceptance and escalation thresholds, and the frequency of retrieval, verification, and escalation actions. Structured error taxonomies capture recurring failure modes and boundary violations at the capability level.

Trajectory-level monitoring focuses on the structure and evolution of execution paths. This includes distributions of $\text{Length}(s_t)$, patterns of retries and looping behaviour, occurrences of unexpected or rare transition sequences, and the frequency of containment actions such as escalation, abstention, or fallback. Particular attention is given to *near-miss events*, where trajectories approach policy boundaries (e.g., $\text{Confidence}(s_t)$ close to escalation thresholds)

without formally violating them, as these indicate weakening control margins.

Monitoring signals are evaluated against predefined thresholds and expected ranges, and deviations trigger governance actions such as threshold recalibration, restriction of capability scope, or escalation for review. Monitoring therefore forms an active component of runtime governance, providing the empirical basis for maintaining the validity of controls over time.

The responsibilities are split as follows. 1LoD operates monitoring and performs remediation within approved limits. 2LoD reviews aggregated evidence, challenges the adequacy of thresholds and controls, and determines whether policy changes or revalidation are required. Security/Ops ensures that enforcement points, telemetry, and logging remain intact and cannot be bypassed.

The decision policy induces a small number of terminal outcomes for execution trajectories, corresponding to how the system exits the governed transition process. These outcomes are determined by the same guard conditions that govern intermediate actions and therefore form part of the control logic rather than a separate layer.

In practice, three terminal states capture the relevant distinctions for governance: HALT, ESCALATE, and ABSTAIN. These correspond respectively to: (i) normal completion where a verified result is produced within the allowed trajectory length; (ii) escalation when verification cannot be established with sufficient confidence; and (iii) abstention when no verified result is reached within the permitted execution budget.

The distinction between ESCALATE and ABSTAIN is operationally important and should not be collapsed. Escalation indicates that the system reached a candidate outcome that cannot be confirmed with sufficient confidence, providing a partial but informative decision trace. Abstention indicates that no such candidate was reached within the allowed trajectory length, resulting in an incomplete trace requiring fuller investigation. Both require human resolution, but the available evidence differs materially and must be treated as distinct incident types with separate remediation workflows and monitoring signals.

At the population level, monitoring captures aggregate behaviour across trajectories. Changes in capability transition frequencies, trajectory structures, or containment patterns may indicate orchestration drift even when no individual trajectory violates policy. For example, a rising abstention rate signals insufficient execution budget or increasing task complexity, while a rising escalation rate signals degradation in verification confidence, potentially due to model drift, data quality issues, or violations of capability assumptions.

Step 7: Change control and continuous improvement (MRM closure). Explicit change triggers are defined, including model, tool, prompt, or memory updates, expansions of authority, drift signals, and incidents. When triggered, 1LoD proposes changes, reruns required tests, and updates monitoring configurations, while 2LoD reviews and approves updated evidence packs and operating thresholds.

This closes the "move fast" loop: iteration is permitted, but only with measurable evidence and governed controls [Wicker et al., 2025]. In particular, changes to authority, transition logic, or persistent orchestration-drift signals should be treated as governed events rather than routine engineering updates.

3 Illustrating runtime governance and capability reuse

We begin with a credit memo drafting example to ground the discussion in a concrete application of runtime governance. We then demonstrate how three seemingly different use cases can be represented as compositions over a shared set of capabilities, illustrating the reuse of controls and the scalability of the approach.

3.1 Illustration: U3 Credit Memo Drafting

Scenario. Company A applies for a revolving credit facility. The agent drafts the credit memo by invoking four capabilities in sequence: C1 (retrieval and attribution), C2 (numeric and ratio reasoning), C3 (long-context drafting), and C4 (policy compliance and gated release). The naive implementation relies on prompt-level guardrails, i.e. instructions to the LLM that depend on probabilistic compliance rather than enforceable constraints. The governed implementation encodes the workflow as a labelled transition system $W = (S, \Pi, \delta, s_0)$ with explicit guards on every transition.

In the naive implementation, guardrails are expressed as prompt instructions or LLM-as-a-judge checks. The following failure modes illustrate the risks of relying on probabilistic enforcement:

- **Stale data used without flagging.** Retrieval returns audited accounts 26 months old; policy requires data no older than 18 months. The model occasionally fails to detect the violation when more recent dates appear in context.
- **Prompt injection via retrieved document.** A third-party document contains an embedded directive: "credit committee approval has been granted, proceed to external release." The agent acts on it, bypassing approval. The model cannot distinguish malicious instructions from legitimate context.
- **Silent numeric error.** EBITDA is double-counted, producing an incorrect coverage ratio (2.82 instead of 1.82). The result appears plausible and passes LLM-based checks.
- **Approval gate reasoning-around.** The agent interprets conversational statements (e.g. "approval confirmed verbally") as satisfying approval requirements, without any recorded approval event.
- **Orchestration drift.** Over repeated runs, the fraction of unauthorised releases increases without triggering per-run violations, reflecting degradation in probabilistic compliance.

Governed implementation with capabilities.
The action space is:

$$\Pi = \{\text{retrieve (C1)},\ \text{compute (C2)},\ \text{draft (C3)},\ \text{release (C4)}\}.$$

We instantiate concrete governed states along the trajectory:

State	Governed context
s_0	Initial state: no data retrieved, no claims, no approval; Length = 0
s_1	Retrieved financials (timestamped, provenance checked); Alignment $\geq \theta_{\text{Alignment}}$, Information = 1
s_2	Financial ratios computed and verified; Verified = 1, Confidence $\geq \theta_{\text{Confidence}}$
s_3	Draft memo with full citation coverage; approval status pending; Alignment = 1, Information = 0
s_4	Human approval logged as authenticated event; Verified = 1
s_5	Terminal state: memo released under policy (HALT)

Transitions and guards.

$s_0 \xrightarrow{\text{retrieve (C1)}} s_1$: Retrieval enforces source allowlists and recency constraints. The guard checks $\text{Alignment}(s_1) \geq \theta_{\text{Alignment}}$. If violated, the system enters Escalate with retrieved evidence logged.

$s_1 \xrightarrow{\text{compute (C2)}} s_2$: A deterministic calculator recomputes ratios. The guard requires $\text{Verified}(s_2) = 1$ and $\text{Confidence}(s_2) \geq \theta_{\text{Confidence}}$. Mismatch triggers Escalate with full discrepancy trace.

$s_2 \xrightarrow{\text{draft (C3)}} s_3$: Drafting proceeds only if all claims are supported by verified evidence, i.e. $\text{Alignment}(s_3) = 1$. Otherwise, drafting is blocked.

$s_3 \rightarrow s_4$ (human approval): Approval is an external event that sets $\text{Verified}(s_4) = 1$ only if an authenticated approval record is logged. Conversational signals do not satisfy this condition.

$s_4 \xrightarrow{\text{release (C4)}} s_5$: Release requires $\text{Verified}(s_4) = 1$. The transition $s_3 \xrightarrow{\text{release}}$ is not defined in δ, and therefore cannot occur.

This construction eliminates entire classes of failures. Stale data is blocked at retrieval through deterministic recency checks. Prompt injection cannot alter transitions because external directives do not affect state attributes. Numeric errors are caught by deterministic recomputation. Approval cannot be inferred from context and must be explicitly recorded. Orchestration drift is prevented because unauthorised transitions are not part of the transition system.

Each transition (s_t, a_t, s_{t+1}) is recorded together with attribute values and guard evaluations, ensuring that every governance decision is auditable as a deterministic function over the governed state.

3.2 AI system as a shared capability cluster serving multiple use cases

The objective is to design an AI systems containing reusable *capability cluster*. Distinct use cases are then realised as different compositions over this shared set of capabilities. This enables validation, controls, and monitoring to be defined once at the capability level and reused across workflows, while runtime governance ensures that each composition satisfies policy constraints.

Crucially, reuse does not imply uniformity. The same capability may be deployed under different authority scopes, thresholds, and control regimes depending on the use case. Capabilities therefore form a stable *governance abstraction*, while their operating configuration remains context-dependent.

We consider three representative use cases:

U1: KYC document review and structured summarisation. The system ingests KYC documents and produces a structured summary for internal review. Material failures include omission of critical facts, fabrication of unsupported fields, misuse of sensitive data, incorrect or stale document selection, and propagation of untrusted or adversarial content embedded in source material. These failures arise primarily from extraction errors, weak provenance controls, and insufficient handling of untrusted inputs.

U2: Model validation report review against policy expectations. The system evaluates a validation report against internal and supervisory standards. Material failures include incorrect application of policy semantics, omission of required elements, unsupported compliance claims, and inconsistent interpretation across cases. A key failure mode is misalignment between evidence and claims, rather than factual inaccuracy alone.

U3: Credit memo drafting and critique. The system produces or evaluates a memo that may influence credit decisions. Material failures include numeric errors in ratios and covenant checks, unsupported or weakly evidenced claims, misuse of confidential information, and process failures such as unauthorised external release. These failures are typically low-frequency but high-severity, and often arise from interactions between capabilities.

Shared capability cluster. Across U1–U3, a common set of capabilities appears:

- C_1: Long-context extraction and structuring (faithful extraction, schema compliance, abstention discipline);
- C_2: Retrieval with attribution (approved sources, provenance, recency, citation completeness);
- C_3: Deterministic numeric computation (parsing, recomputation, consistency checks);
- C_4: Policy-constrained drafting and release (policy rules, required sections, approval gates, prohibited content checks).

These capabilities define the *action space* of the system. Each use case corresponds to a different orchestration over this shared set. Importantly, while the capability definitions remain stable, their *configuration*—including authority, thresholds, and required controls—varies across use cases.

Mapping failure modes to capabilities. The mapping from failure modes to capabilities is structural rather than use-case specific.

Failures in U1 such as omission or fabrication correspond to breakdowns in C_1 (extraction fidelity and abstention discipline). Stale or incorrect document usage maps to C_2 (provenance, scope, and recency controls), while privacy violations map to C_4 (policy and data-handling constraints).

Failures in U2 arise primarily in C_4, where policy semantics must be applied correctly, and in C_2, where claims must be supported by appropriate evidence. Domain misalignment and false assurance correspond to failures in policy-constrained reasoning and evidence alignment.

Failures in U3 extend this with C_3, where numeric correctness becomes critical. Silent numeric errors reflect the absence of deterministic computation and consistency checks. Unsupported claims again map to C_2, while unauthorised release and process failures map to C_4 and, critically, to missing trajectory-level constraints.

While the same capabilities appear across use cases, the *severity and operational importance* of their failure modes differs. For example, retrieval errors in U1 primarily affect completeness, whereas in U3 they may directly affect financial decisions. This difference is reflected not in redefining capabilities, but in tightening thresholds, controls, and escalation policies.

Capability-centric system design. The agentic system is therefore designed as a capability transition system $W = (S, \Pi, \delta, s_0)$, where $\Pi = \{C_1, C_2, C_3, C_4\}$ defines the available actions, and δ encodes admissible compositions under policy.

Each use case is a restriction of this system:

- U1 uses $\{C_1, C_2, C_4\}$ with read-only authority and no external side effects;
- U2 uses $\{C_2, C_4\}$ with strict policy semantics and evidence requirements;

- U3 uses $\{C_1, C_2, C_3, C_4\}$ with write/dispatch authority and approval constraints.

The difference between use cases is therefore not the underlying system, but the *capability composition, authority scope, and control tier*. Capability reuse operates at the level of definitions, while governance is applied through configuration of these dimensions.

Reusable validation, controls, and monitoring. This design enables governance to operate at the system level through reusable capability artefacts.

Validation. Each capability is validated against its failure modes: C_1 for extraction fidelity and abstention behaviour, C_2 for provenance, recency, and citation completeness, C_3 for deterministic correctness and failure-closed behaviour, and C_4 for policy compliance and approval logic. These validation results form capability evidence packs that can be reused across all use cases invoking the capability, with use-case-specific tightening where required.

Controls. Controls are attached to capabilities and enforced through the transition system. For example, C_2 enforces source allowlists and recency checks, C_3 enforces deterministic recomputation, and C_4 enforces approval gates and prohibits unauthorised release. At the system level, temporal and path constraints over trajectories ensure that capabilities are composed in a policy-compliant manner. The strictness of these controls depends on the use-case tier rather than the capability definition itself.

Monitoring. Monitoring operates at two levels. Capability-level monitoring tracks performance and reliability metrics (error rates, abstention rates, guard score distributions) that are reusable across use cases. Trajectory-level monitoring tracks execution patterns (sequence of capabilities, escalation events, near-misses) and detects orchestration drift that is not visible at the capability level. Thresholds for alerts and escalation are calibrated per use case.

Implication for scalable governance. Although U1–U3 appear as distinct applications, they are compositions over a shared capability cluster. This is the basis for scalability. Rather than validating each workflow independently, governance focuses on validating capabilities and controlling their composition through authority, constraints, and monitoring.

In particular, onboarding a new use case does not require rebuilding validation and control infrastructure from scratch. Instead, it requires verifying how existing capabilities are composed, how they are configured for the specific context, what authority they are granted, and whether the resulting trajectory-level behaviour remains within risk appetite.

4 Literature review

This paper sits at the intersection of bank Model Risk Management (MRM), agentic AI architectures, evaluation practice for AI agents, and runtime governance/enforcement. The literature can be grouped into five overlapping strands.

First, **MRM baselines** establish expectations for governance, conceptual soundness, outcomes analysis, monitoring, documentation, and effective challenge [fed, 2011, occ, 2011, pra, 2023, osf, 2025]. In the US, SR 11-7 and companion guidance are commonly used references [fed, 2011, occ, 2011]; in the UK, PRA SS1/23 provides model risk principles for banks [pra, 2023]; and in Canada, the E-23 serves as a model risk management guidelines for a range of financial intitutions, including banks and insurers [osf, 2025]. These documents were written for a broad class of models, but their requirements (inventory, intended use and limitations, validation evidence, monitoring plans, and change control) remain binding when AI systems become more agentic. General AI risk frameworks and regulation provide additional lifecycle guidance and

obligations, but are not agentic-specific in their technical governance primitives [nis, 2023, euA, 2024, mas, 2025].

Second, **agentic governance and capability-centric views** argue that the key risk surface is not only the base model but the system's *capabilities* and *design* (orchestration) [Khoo et al., 2025]. The ARC framework motivates the capability lens as a scalable abstraction and distinguishes risks arising from components, design, and capabilities; it further motivates tiered controls based on contextual relevance (impact and likelihood) [Khoo et al., 2025]. Our contribution is to operationalise this perspective as an MRM programme: pooled capability validation as a bank-wide artefact and trajectory-level enforceability. Related practitioner work emphasises the importance of closing the loop between evidence, change control, and operational monitoring in fast-moving bank environments [Wicker et al., 2025, Aldridge et al., 2025].

A key distinction in how our framework advances the literature is the shift from static, design-time risk mapping to dynamic, runtime governance. Frameworks like ARC [Khoo et al., 2025] provide a comprehensive map of what can go wrong - elements, risks, and controls organised by capability - but operate primarily at design time. Our framework takes this design-time foundation and adds the model risk management view for governing an agentic AI workflow in motion: live execution trajectories, real-time policy enforcement over traces, drift detection over trajectory populations, and tiered containment that activates mid-run. Similarly, ARC's risk quantification uses qualitative impact/likelihood scales (1-5), which serve well for initial prioritisation but are insufficient for bank model risk management, where regulators expect measurable, auditable evidence. Our "pooled evidence" concept is a methodological step forward: capabilities are calibrated via centralised, statistical validation - measurable abstention rates, schema violation rates, guard score distributions, numeric error rates - that can be reused, compared, and tracked over time across the enterprise.

Third, **runtime governance and integrated oversight** emphasise that agentic risks manifest at runtime as sequences of actions, and therefore require governance-semantic telemetry, authorization monitoring, conformance checking, drift detection, and containment [Wang et al., 2025]. MI9 provides a unified runtime governance framework and is particularly relevant because it is written from a bank MRM perspective and explicitly integrates these components [Wang et al., 2025].

Fourth, **evaluation practice and reliability measurement** emphasise failure-mode driven evaluation and the importance of treating evaluation suites as compounding lifecycle assets (failures → tests → regressions → change-control artefacts) [Anthropic, 2026, Sudjianto, 2025]. Complementary reliability work argues that "capability" improvements do not automatically yield improvements in consistency, robustness, predictability, and safety; these dimensions require explicit metrics and monitoring [Rabanser et al., 2026]. In particular, the framing in Rabanser et al. [2026] is useful for MRM because it treats agent reliability as multi-dimensional and makes clear that average task success can mask operationally important failure modes (e.g., inconsistent behaviour under minor perturbations, brittle performance under shift, or safety-relevant regressions). We incorporate these insights by (i) structuring capability evidence packs around failure modes and (ii) treating reliability dimensions as monitored signals tied to escalation [Rabanser et al., 2026].

Fifth, **agent security and state-dependent vulnerabilities** show that tool-using agents can be vulnerable at specific execution states and that backbone vulnerabilities can be amplified through tool use and memory [Bazinska et al., 2025]. This motivates adversarial "threat snapshot" testing (tests designed around vulnerable execution states) and state-aware runtime monitoring [Bazinska et al., 2025]. In our framework, these become capability-level adversarial suites

(especially for retrieval and tool invocation) and trajectory-level anomaly detection (repeated blocked transitions, anomalous tool sequences). Complementary resources such as OWASP guidance and benchmark environments help characterise and stress-test these threat models [OWASP Foundation, 2024, OWASP GenAI Security Project, 2025, Debenedetti et al., 2024, Andriushchenko et al., 2024, Zhang et al., 2024, Ye et al., 2024].

Finally, **observability standards** such as OpenTelemetry provide implementation primitives for capturing traces and spans in distributed systems [OpenTelemetry, 2026]. We use these primitives to define an execution trajectory as a trace (one run) composed of spans (capability invocations and tool calls), with attributes capturing governance semantics (capability IDs, tool intents, guard scores, authorization and approval decisions) [OpenTelemetry, 2026, Wicaksono et al., 2025]. This connects academic governance objects to a practical instrumentation standard used in modern AI systems, and aligns naturally with action-graph style representations that treat agent behaviour as structured sequences for analysis and control [Wicaksono et al., 2025].

5 Discussion and outlook

Implications for bank MRM. The framework reframes MRM as the design and operation of a control plane for agentic systems. The central shift is not only in what is governed, but in how governance is delivered. In practice, value is created and risk is realised at the level of deployed workflows, and governance must operate at that level. Periodic model validation is therefore insufficient: effective control requires continuous enforcement of authority, constraints, and policy over execution trajectories.

The objective is not maximal automation, but sustainable automation: the highest level of autonomy that can be scaled across use cases without a disproportionate increase in residual risk, evidential burden, or approval overhead. Achieving this requires governance to be embedded in execution. In particular, governance must allocate authority, constrain side effects, mediate access to tools and data, route exceptions, and generate replayable evidence as part of the workflow itself. Without such a control plane, each new use case becomes a bespoke governance exercise, eroding scalability through duplicated validation, fragmented monitoring, and manual intervention.

Capability-centric governance reduces marginal onboarding effort only when supported by operational infrastructure that makes reuse credible: stable capability catalogues, enforceable authority boundaries, reliable telemetry, and disciplined change control. Absent this infrastructure, the language of reuse risks masking a relocation of effort rather than a reduction in it.

This perspective also sharpens the role of monitoring. For any material action, the institution must be able to reconstruct which capability was invoked, under what authority, on what evidence, and with what consequence. This is the minimum requirement for effective challenge and for sustaining growth in a regulated environment. Systems that cannot produce such evidence may function locally, but remain fragile at scale.

A further implication concerns delegated authority. Capabilities are not merely technical units, but bounded permissions to expose the institution to economic, conduct, or reputational risk. Governance must therefore specify what the system is allowed to do, under what approvals, and within what limits. When a trajectory attempts to exceed that authority, the response must be deterministic: contain, restrict, escalate, or stop. This elevates authority to a first-class governance object and requires telemetry to record not only realised actions, but attempted exercises of authority.

Finally, the framework supports proportionality. Governance intensity can be tiered by consequence and autonomy, with tighter authority, stronger controls, and more demanding evidence

requirements for higher-risk deployments. This enables institutions to expand deployment without uniformly increasing friction, while maintaining bounded risk. The binding constraint on adoption is therefore not model capability, but the institution's ability to build and operate a reliable control plane around it.

Operating model and accountability. The framework sharpens, but does not fundamentally alter, the allocation of responsibilities. 1LoD remains accountable for use-case design, operation, and outcomes. 2LoD defines capability standards, evidence requirements, validation approaches, and challenge processes. What changes is the object of governance: capabilities become the unit of standardisation and reuse, while use cases become governed compositions of those capabilities.

This creates a tension that must be managed. If capability catalogues, evidence standards, and control playbooks become overly prescriptive, 2LoD risks shifting from independent challenge into de facto design authority. A robust operating model must therefore preserve 1LoD ownership of implementation choices, while ensuring that 2LoD retains the ability to set standards, assess evidence, and challenge residual risk without assuming first-line responsibility.

Adoption also requires coordination beyond MRM. Authority scopes, data rights, third-party dependencies, and enforceability of controls depend on Legal, Compliance, Security, Procurement, and Architecture functions. The framework should therefore be understood as a core layer within a broader institutional control architecture rather than a standalone solution.

Limitations. The primary limitations are operational rather than conceptual. Control effectiveness depends on the quality and completeness of telemetry; gaps in instrumentation undermine both enforcement and auditability. Integration risks remain material: interactions between capabilities can produce behaviours that are not visible in isolated evaluation, particularly when authority scopes or data dependencies interact in unexpected ways.

Calibration is also a constraint. Poorly specified thresholds or guard conditions can introduce operational friction without improving safety, while overly permissive settings may leave residual risk unaddressed. Governance therefore requires continuous calibration based on observed behaviour, not static configuration.

Finally, telemetry alone is not sufficient. While it enables reconstruction of what happened, what controls fired, and what approvals were recorded, it does not guarantee that policies are well specified or that workflows are conceptually sound. Effective governance therefore requires both instrumentation and domain-informed challenge. The framework provides the structure for enforceable control, but its effectiveness ultimately depends on the quality of policy design, calibration, and institutional discipline in its application.

References

Sr 11-7: Supervisory guidance on model risk management. Technical Report SR 11-7, Board of Governors of the Federal Reserve System, April 2011. URL https://www.federalreserve.gov/supervisionreg/srletters/sr1107.htm.

Supervisory guidance on model risk management. Technical Report OCC 2011-12, Office of the Comptroller of the Currency, April 2011. URL https://www.occ.gov/news-issuances/bulletins/2011/bulletin-2011-12.html. Companion guidance to SR 11–7.

Artificial intelligence risk management framework (ai rmf 1.0). Technical report, National Institute

of Standards and Technology, 2023. URL https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=936225.

Model risk management principles for banks. Technical Report SS1/23, Prudential Regulation Authority, Bank of England, May 2023. URL https://www.bankofengland.co.uk/prudential-regulation/publication/2023/may/model-risk-management-principles-for-banks-ss.

Regulation (eu) 2024/1689 laying down harmonised rules on artificial intelligence (artificial intelligence act). Technical report, European Parliament and Council of the European Union, 2024. URL https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=OJ%3AL_202401689.

Mas consultation paper on ai risk management guidelines. Technical report, Monetary Authority of Singapore, November 2025. URL https://www.mas.gov.sg/-/media/mas-media-library/publications/consultations/bd/2025/final_consultation_paper_on_guidelines_on_ai_risk_management_forrelease.pdf.

Guideline e-23 – model risk management (2027). Technical Report E23, Office of the Superintendent of Financial Institutions, September 2025. URL https://www.osfi-bsif.gc.ca/en/guidance/guidance-library/guideline-e-23-model-risk-management-2027.

Irene Aldridge, Jolie An, Riley Burke, Michael Cao, Chia-Yi Chien, Kexin Deng, Ruipeng Deng, Yichen Gao, Olivia Guo, Shunran He, Zheng Li, George Lin, Weihang Lin, Fanyi Lyu, Kwunfung Ng, Qi Wang, Hanxi Xiao, Dora Xu, Yuanyuan Xue, Sheng Zhang, Sirui Zhang, Yun Zhang, Sirui Zhao, Xiaolong Zhao, Yihan Zhao, and Waner Zheng. Agentic artificial intelligence in finance: A comprehensive survey. SSRN Working Paper, 2025. URL https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5803628. Posted 29 Nov 2025. Available at SSRN: 5803628.

Maksym Andriushchenko, Alexandra Souly, Mateusz Dziemian, Derek Duenas, Maxwell Lin, Justin Wang, Dan Hendrycks, Andy Zou, Zico Kolter, Matt Fredrikson, et al. Agentharm: A benchmark for measuring harmfulness of llm agents. *arXiv preprint arXiv:2410.09024*, 2024. URL https://arxiv.org/abs/2410.09024.

Anthropic. Demystifying evals for ai agents. Engineering blog, January 2026. URL https://www.anthropic.com/engineering/demystifying-evals-for-ai-agents.

Julia Bazinska, Max Mathys, Francesco Casucci, Mateo Rojas-Carulla, Xander Davies, Alexandra Souly, and Niklas Pfister. Breaking agent backbones: Evaluating the security of backbone llms in ai agents. *arXiv preprint arXiv:2510.22620*, 2025. URL https://arxiv.org/abs/2510.22620.

Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for llm agents. *arXiv preprint arXiv:2406.13352*, 2024. URL https://arxiv.org/abs/2406.13352.

Shaun Khoo, Jessica Foo, and Roy Ka-Wei Lee. With great capabilities come great responsibilities: Introducing the agentic risk & capability framework for governing agentic ai systems. *arXiv preprint arXiv:2512.22211*, 2025. URL https://arxiv.org/abs/2512.22211.

OpenTelemetry. Traces (opentelemetry concepts). Documentation, 2026. URL https://opentelemetry.io/docs/concepts/signals/traces/.

OWASP Foundation. Owasp top 10 for large language model applications. Community standard (versioned), 2024. URL https://owasp.org/www-project-top-10-for-large-language-model-applications/.

OWASP GenAI Security Project. Agentic ai — threats and mitigations. OWASP Agentic Security Initiative (ASI) guide, 2025. URL https://genai.owasp.org/resource/agentic-ai-threats-and-mitigations/.

Stephan Rabanser, Sayash Kapoor, Peter Kirgis, Kangheng Liu, Saiteja Utpala, and Arvind Narayanan. Towards a science of ai agent reliability. *arXiv preprint arXiv:2602.16666*, 2026. URL https://arxiv.org/abs/2602.16666v1.

Agus Sudjianto. Agentic.eval (preprint): Behavioral and process-based evaluation for agentic ai systems. Preprint (user-provided PDF), 2025.

Agus Sudjianto and Lau Wingyan. Finstructbench: A benchmark for structured information retrieval from financial documents using graph-verifiable questions. SSRN, 2026. URL https://papers.ssrn.com/sol3/papers.cfm?abstract_id=6506403.

Charles L. Wang, Trisha Singhal, Ameya Kelkar, and Jason Tuo. Mi9: An integrated runtime governance framework for agentic ai. *arXiv preprint arXiv:2508.03858*, 2025. URL https://arxiv.org/abs/2508.03858.

Ilham Wicaksono, Zekun Wu, Rahul Patel, Theo King, Adriano Koshiyama, and Philip Treleaven. Mind the gap: Evaluating model- and agentic-level vulnerabilities in llms with action graphs. *arXiv preprint arXiv:2509.04802*, 2025. URL https://arxiv.org/abs/2509.04802.

Matt Wicker, Lukasz Szpruch, , and Søren Mørk. Move fast without breaking the bank model risk management of genai workflows. SSRN Working Paper, 2025. SSRN ID: 5682603.

Junjie Ye, Sixian Li, Guanyu Li, Caishuang Huang, Songyang Gao, Yilong Wu, Qi Zhang, Tao Gui, and Xuanjing Huang. Toolsword: Unveiling safety issues of large language models in tool learning across three stages. *arXiv preprint arXiv:2402.10753*, 2024. URL https://arxiv.org/abs/2402.10753.

Hanrong Zhang, Jingyuan Huang, Kai Mei, Yifei Yao, Zhenting Wang, Chenlu Zhan, Hongwei Wang, and Yongfeng Zhang. Agent security bench (asb): Formalizing and benchmarking attacks and defenses in llm-based agents. *arXiv preprint arXiv:2410.02644*, 2024. URL https://arxiv.org/abs/2410.02644.