Distributing Accountability, Not Capability

Phase Separation and the LLM Workflow Quadrant in Autonomous AI Agent Architectures

Author: Tatsuya Shimomoto **Affiliation:** Independent researcher **ORCID:** 0009-0002-6168-4162 **Version:** v1 (2026-05-23) **DOI:** 10.5281/zenodo.20353789 (Zenodo — concept DOI, resolves to the latest version) **Companion repository:** Agent Attribution Practice (AAP), DOI 10.5281/zenodo.20251893 (v0.3.0)

Abstract

Autonomous AI agents in business deployments exhibit a recurring failure mode: when an incident occurs, responsibility cannot be redirected to a separable contributor. The dominant discourse treats this as a single phenomenon, addressed by sandboxing, human-in-the-loop overload, or what Elish (2019) named the *moral crumple zone*. This paper argues the phenomenon is two architecturally distinct failure modes that have been conflated, and that the conflation is sustained by a missing positive name and a missing time-axis.

The paper introduces two contributions. First, a four-quadrant decomposition of business AI work — along the axes of *deterministic vs semantic-judgment* and *pre-defined vs exploratory* — yields a positive name for the cell most current LLM applications occupy: the *LLM Workflow Quadrant*. The quadrant is defined by a single load-bearing property: the path is decided in advance by humans or by code, and the LLM is called as a single bounded step within that path; the property divides naturally into a conversational sub-form (specialized chat agents) and a batch sub-form (single-purpose LLM functions inside deterministic pipelines). The decomposition distinguishes *principled* from *artificial redirect impossibility*: the former intrinsic to autonomous loops, the latter the product of routing workflow work through autonomous-loop architecture by elimination, with four downstream symptoms (the RPA exception-handling bottleneck, the sandbox-strength demand, the structural distortion of human-in-the-loop, and the dissolution of the accountability chain at postmortem). Second, a *Phase Separation* axis (design vs operation), independent of Quadrant, surfaces a *Phase-crossing decision* — recorded at deployment time, in one sentence — required when an autonomous-loop component is placed in the operation phase. The Phase axis descends recursively to skill-design granularity, where the Quadrant 3 ↔ Quadrant 4 boundary is a continuous gradient on which model capability is downstream of phase, not the primary lever.

The consequence is procedural rather than architectural: deployments make the Phase-crossing decision explicit, designate a *pre-named gap-bearer* for principled-impossibility placements, and route artificial-impossibility cases to re-architecture. The framework complements existing AI risk-management and management-system standards by recording the *judgment layer* they presuppose. Both rules are stated as experimental; the open questions are the research agenda.

Keywords: AI accountability; autonomous agents; attribution gap; LLM workflow; phase separation; AI governance; moral crumple zone.

---

1. Introduction

Three years into the deployment of large-language-model agents in business systems, a recurring failure mode has come into focus. When an autonomous agent acts on the world — sends a message, signs a document, allocates capital, deletes a record — and the act turns out to be wrong, the operator cannot redirect responsibility to a separable contributor. The model output, the tool selection, the retrieved context,

and the prompt assembly blend into a single stream of judgments at runtime. After the incident, there is nothing to point at. The failure dissolves across the architecture.

The dominant discourse treats this dissolution as a single phenomenon. Sandboxing aims to bound its blast radius. Human-in-the-loop reviews aim to catch its mistakes before they escape. Elish (2019), in the most-cited treatment of the pattern, named it the *moral crumple zone*: a structural arrangement in which the responsibility of an autonomous system is pushed onto a human operator whose actual control over the runtime decisions is limited. Each of these responses is locally reasonable. None of them surfaces that the discourse is collapsing two architecturally distinct failures into one name.

This paper argues that the collapse is sustained by two missing pieces of vocabulary. The first is a *positive name* for the design space in which most current LLM applications actually live — a space the industry has been routing through autonomous-loop patterns by elimination because no positive name exists. The second is a *time-axis* in the discourse: a distinction between the *design phase* of a deployment, in which a workflow is being discovered and built, and the *operation phase*, in which a deployed workflow runs on its known path. The two phases optimize for opposite properties — *flexibility* in design, *predictability* in operation — and conflating them produces a recurring class of architectural choices whose accountability cost is paid only after the fact.

The argument is offered in three layers. The *surface* layer is misapplication: workflow work routed through autonomous-loop architecture, producing an attribution gap that is in fact resolvable by re-architecture. The *middle* layer is the vocabulary gap that sustains the misapplication: in the absence of a positive name for "semantic-judgment with a known path," every semantic-judgment task gets routed to "autonomous loop" by default. The *deep* layer is the phase conflation: the assumption that an "always-on autonomous agent runs the business" is a coherent target, achievable by capability improvement, rather than a category that compresses two opposite optimization axes into one runtime.

The paper draws on a sister body of work: the implementation and operational history of a single LLM-based agent, recorded as a series of architecture decision records in a public repository (Shimomoto 2026), and a sequence of essays from which the decision records were extracted. The argument is presented here as a *position*, not as an empirical study. The repository is the empirical substrate; the seven-essay narrative spine (Shimomoto 2026, Articles 1–7) is the discovery trajectory; this paper distills the architectural follow-up (Articles 4–7) into a single, harness-neutral statement.

The remainder is organized as follows. Section 2 introduces the four Business AI Quadrants. Section 3 distinguishes principled from artificial redirect impossibility. Section 4 names the LLM Workflow Quadrant vocabulary gap. Section 5 introduces Phase Separation. Section 6 records the Phase-crossing decision. Section 7 descends the Phase axis to skill-design granularity. Section 8 situates the framework against existing risk-management and management-system standards. Section 9 closes with open questions.

Figure 1 — A three-layer diagnosis. Surface: workflow work misapplied through autonomous-loop architecture (Section 3). Middle: vocabulary gap that routes by elimination (Section 4). Deep: phase conflation that compresses design and operation into one runtime (Sections 5–7).

The thesis the paper carries throughout is that what organizations have spent centuries refining is not the distribution of *capability*, but the distribution of *accountability*. The current AI-safety discourse, dominated by capability-axis interventions (more capable models, stronger sandboxes, larger human-in-the-loop budgets), addresses what an agent *can do*. It does not address *to whom what an agent has done can be redirected*. Where this paper distinguishes them, *attribution* names the technical property of post-

hoc separability — whether a contribution can be redirected — and *accountability distribution* names the organizational pattern that depends on attribution succeeding; an *attribution gap* is the foreclosure of the former, which then forecloses the latter. The framework below is one path back to that question.

---

2. The Four Business AI Quadrants

Business AI work decomposes along two axes that are orthogonal to agent architecture: the *control structure* of the work and the *path stability* of the work.

The first axis, *control structure*, distinguishes between work in which each step is fully determined by code (the next operation is computable from the current state by a deterministic procedure) and work in which the next operation requires *semantic judgment* over natural-language input (the choice cannot be expressed as a deterministic function over typed inputs). The first regime is the domain of classical software engineering; the second is the regime that motivated the introduction of LLMs into business systems in the first place.

The second axis, *path stability*, distinguishes between work whose execution path is known at design time (the steps and their connections are enumerable in advance) and work whose execution path is *exploratory* — discoverable only at runtime, dependent on what the input turns out to require. The first regime is amenable to flowcharts, state diagrams, and pipelines; the second is the regime in which the next step depends on the previous result in a way that cannot be foreseen.

The cross-product yields four quadrants:

	Pre-defined workflow	Exploratory
Deterministic	(1) Script Quadrant	(2) Algorithmic Search Quadrant
Semantic-judgment	(3) LLM Workflow Quadrant	(4) Autonomous Agentic Loop Quadrant

(1) The **Script Quadrant** is the domain of forms, normalizers, pipelines, lookups, and validators. The accountability story is the classical software-engineering one: each step has named code, named authors, and version control. (2) The **Algorithmic Search Quadrant** is the domain of constraint solvers, mixed-integer programming, classical search, scheduling, and combinatorial allocation. The accountability story is the algorithm-engineering one: heuristics and objective functions are designed, parameterized, and validated against known cases.

The two semantic-judgment quadrants are the contested terrain. (3) The **LLM Workflow Quadrant** is the design space in which *the execution path is decided in advance — by humans, by code, by the surrounding workflow — and the LLM is called as a single, bounded step within that path* to perform a semantic judgment whose role is fixed by the path. The LLM does not decide the next action; the next action is already decided by the calling pipeline or by the human operator running the session. The LLM's output fluctuates probabilistically — the same input does not always return exactly the same output — but the *role* the LLM plays at each call is fixed by the surrounding path. As a consequence — not as part of the definition — the contribution of each call is post-hoc separable. This single quadrant divides naturally into two sub-forms by input/output modality:

(3a) **Conversational sub-form**: specialized chat agents that pair retrieval, a system prompt, and (where needed) conversation history with bounded LLM calls. Examples include legal-consultation assistants, diagnostic-support assistants, internal-FAQ systems, and expert-knowledge support tools. The human in

the conversation is the *judging agent*; the LLM contributes knowledge retrieval and organization. Multi-turn conversations that narrow conditions or differentiate diagnoses can still sit in this sub-form when each turn's LLM call has a documented role and the human, rather than the LLM, decides what to do next.

(3b) **Batch sub-form**: an ordinary codebase whose flow control is written in conventional code (Python in the examples below, but the architecture is language-agnostic) and whose semantic-judgment leaves are handled by *LLM functions*. This paper introduces the **LLM function** as the architectural primitive of (3b): an ordinary function in the codebase whose body delegates the judgment to an LLM call, with a defined input type, a defined output schema, and one judgment responsibility. From the caller's perspective the LLM function behaves like any other function — defined input goes in, a value of a defined type comes out — except the output may fluctuate probabilistically since an LLM is the judging organ. The surrounding codebase owns control flow; LLM functions occupy only the leaves the codebase cannot decide deterministically. Crucially, the Batch sub-form does *not* require — and is, in shape, incompatible with — a general-purpose agent: each judgment category gets its own narrow LLM function, and fifty judgment categories means fifty narrow functions rather than one generalist.

Two concrete examples ground the architecture. In both, the system is ordinary code with LLM functions invoked inline.

Invoice matching. A function that decides whether to approve, reject, or escalate an invoice runs mostly on deterministic checks — purchase-order lookup, expiry comparison, duplicate detection by hash, percentage-tolerance amount check. About 80% of invoices route on these alone. The remaining ones reach a single semantic-judgment branch — the totals do not match, but the line items may be worded differently while still corresponding — where the code calls an LLM function:

```python
def process_invoice(invoice: Invoice) -> Action:
    po = lookup_po(invoice.po_id)
    if po is None:
        return Action.REJECT_NO_PO
    if invoice.date > po.expiry:
        return Action.REJECT_EXPIRED
    if is_duplicate(invoice):
        return Action.REJECT_DUPLICATE
    if abs(invoice.amount - po.amount) / po.amount <= 0.01:
        return Action.APPROVE

    # one semantic-judgment leaf, called as a function:
    verdict = match_line_items(invoice.lines, po.lines)  # ← LLM function
    if verdict == "MATCH":
        return Action.APPROVE
    elif verdict == "PARTIAL":
        return Action.ESCALATE_FOR_HUMAN_REVIEW
    else:
        return Action.REJECT_AMOUNT_MISMATCH
```

`match_line_items(invoice_lines, po_lines) -> Verdict` is an ordinary Python function whose body happens to be an LLM call: it accepts a defined input, returns a verdict in a fixed schema (`MATCH` / `PARTIAL` / `NO_MATCH`), and carries no other responsibility. The function does not decide what to do next; the next step is encoded in the surrounding `if`/`elif`/`else` block.

Comment generation in Contemplative Agent. The author's *Contemplative Agent* (Shimomoto 2026, companion sibling repository) runs a feed-processing pipeline that decides whether to engage with each post and, if so, generates a comment grounded in a stated constitution of values. The pipeline body is ordinary

Python control flow that intersperses deterministic gates (per-author 24-hour rate limits, processed-post deduplication, write-budget guards) with LLM function calls at the semantic-judgment points:

```
def should_engage_with_post(post_text, post_id, author_id, ctx):
    if already_processed(post_id): return False
    if author_rate_limited(author_id, ctx): return False

    # LLM function: post text → relevance score in [0.0, 1.0]
    score = score_relevance(post_text)

    threshold = domain_threshold_for(author_id, ctx)
    if score < threshold:
        if score >= UPVOTE_ONLY_THRESHOLD:
            client.upvote_post(post_id)
        return False
    return True  # downstream code will call generate_comment

def create_comment(post_text):
    comment = generate_comment(post_text)  # ← LLM function
    if comment is None: return None
    if is_duplicate(comment): return None  # deterministic dedup
    return comment
```

`score_relevance(post_text) -> float` and `generate_comment(post_text) -> Optional[str]` are LLM functions in the same sense as `match_line_items`: each takes a defined input, returns a value of a defined type, and carries one judgment responsibility. The pipeline decides the engagement workflow — when to upvote, when to skip, when to comment, when to deduplicate — and the LLM functions occupy only the semantic-judgment leaves. No agent loop sits between them; the architecture has the same shape as the invoice-matching code.

In both sub-forms the load-bearing property is the same: *the path is decided in advance, the LLM serves a bounded role within it.* Post-hoc separability is the consequence — the operator can identify which call produced which contribution and route the consequence accordingly — but the essence is the structural decision that the LLM is not the one deciding the next action. The probabilistic fluctuation of any single call does not break the property, because exception-judgment work was probabilistic when humans did it too; what the LLM Workflow Quadrant preserves is not deterministic per-call output but the *fixed role* of each call within a path the humans have specified.

(4) The **Autonomous Agentic Loop Quadrant** is the architecture in which the LLM decides each next step at runtime — the ReAct pattern of Yao et al. (2022) and its descendants. The model output, tool selection, history reference, and prompt context blend into a single stream of judgments. Each iteration's role is decided dynamically; the decomposition that produced the iteration is not recoverable from outside.

The quadrants are *not agent categories*. The same agent process can be deployed against work in different quadrants; what changes is the architecture chosen for the work. The quadrants describe the *work*, not the agent. This distinction is load-bearing: it is the work — its control structure and its path stability — that determines whether accountability can be distributed across the resulting system. Capability-axis improvements (better models, longer context windows, smaller hallucination rates) do not move work between quadrants. A piece of work whose path requires runtime exploration is not made non-exploratory by a more capable model; it is made cheaper to fail, but the failure is still principled.

Figure 2 — The 2 × 2 Business AI Quadrants. Each cell carries a named architecture and an accountability story. Quadrant 3 (LLM Workflow) divides by I/O modality into a conversational sub-form (specialized chat agents) and a batch sub-form (deterministic pipelines that call LLM functions — single-purpose, schema-bounded LLM calls — at their semantic-judgment leaves). Anthropic (2024) and OpenAI (2025) document composition patterns that operate in Q3 territory without naming the cell positively; the ReAct loop of Yao et al. (2022) is the canonical Quadrant 4 architecture.

The Quadrants serve in this paper as the diagnostic frame within which redirect-impossibility is distinguished. The next section uses the frame to isolate two failure modes that are routinely treated as one.

---

3. Two Redirect Impossibilities

When an autonomous AI system fails in production, the question that determines whether accountability can be distributed is *to whom can responsibility be redirected*. The operator inspects the failure, identifies its locus, and routes the consequence — managerial, technical, contractual, legal — to the party whose contribution produced it. *Redirect* is the act of routing. *Distribution* is what redirect produces when it succeeds.

In two architecturally distinct configurations, redirect does not succeed. The first configuration is the one most accountability discourse addresses; the second is the one most accountability discourse misses.

Artificial redirect impossibility. Work that belongs in the LLM Workflow Quadrant has been implemented through Autonomous Agentic Loop architecture. The work itself — say, a contract-review pipeline whose path is enumerable in advance, with bounded LLM calls at each named step, or an internal-FAQ chat agent whose turns combine retrieval and a system-prompted LLM call under a human's directional control — would tolerate post-hoc separation: a failure in clause classification could be redirected to the classifier's prompt designer, a failure in jurisdiction inference could be redirected to the jurisdiction-lookup configurator, a failure in retrieval grounding could be redirected to the corpus owner. Implemented as an autonomous loop, the same work loses separability: the loop's output blends classification, lookup, retrieval, and routing into one stream of judgments. The redirect target dissolves. The impossibility is *artificial* in the strict sense that the architecture, not the work, foreclosed it. Re-architecting the same work as an LLM Workflow Quadrant component — in its conversational sub-form for the chat agent, in its batch sub-form for the pipeline — restores separability.

Principled redirect impossibility. Work that genuinely requires open-ended exploration — research assistance over an unbounded corpus, exploratory data analysis whose next step depends on what the current step found, novel-problem coding whose plan emerges as the problem is understood — sits in the Autonomous Agentic Loop Quadrant by structural necessity. The runtime blending that forecloses redirect is *intrinsic* to the architecture; it is what makes the architecture capable of doing the work in the first place. Re-architecting this work as a workflow would prevent it from being done at all. The impossibility is *principled* in the strict sense that it is not architecturally resolvable: it is a property of the work as much as of the architecture.

The two cases call for opposite remedies. The artificial case calls for *re-architecture*: the work is moved to the LLM Workflow Quadrant, where its bounded calls and named roles restore separability. The principled case calls for *commitment*: the deployment designates a *pre-named gap-bearer* — an organizationally identifiable *human* who absorbs the attribution gap as a deliberate, recorded cost, named before deployment rather than discovered after an incident; not a team, not the LLM itself, not a post-hoc assignment — and accepts that failure analysis will identify *that* the system failed, but will not produce a clean redirect target.

The two cases have been routinely conflated. The literature on the *moral crumple zone* (Elish 2019) describes the symptom of conflation: when an autonomous system fails and there is no pre-named gap-bearer, the burden falls on the operator with the least actual control — the most recent engineer, the on-call rotation, the user. Elish's account does not distinguish the artificial case from the principled case, because the failure pattern looks the same from the postmortem side. From the architectural side, the two are not the same at all: one calls for moving work back to a different quadrant; the other calls for a structural commitment at deployment time.

The artificial case has a recognizable downstream signature in production. Four symptoms, routinely discussed as separate problems, share this root. (i) The *RPA exception-handling bottleneck*: organizations that mechanized Quadrant 1 work with robotic process automation found that the remaining work — semantic-judgment exception handling — accumulated as manual overhead, because RPA had no vocabulary for Quadrant 3 and could not factor it out. The same bottleneck reappears in the agent era under a new name when Quadrant 3 work is again routed wholesale to autonomous loops. (ii) The *sandbox-strength demand*: autonomous loops over what is structurally Quadrant 3 work need maximum isolation — process isolation, microVMs, WebAssembly sandboxes — because the loop's runtime behavior is not predictable from outside; the same work decomposed into bounded-role functions could have permission boundaries designed at the business-task level. (iii) The *structural distortion of human-in-the-loop*: what the term "HITL" advertises as a feedback loop in which humans improve the AI functions in practice as a structure in which humans continuously absorb the Quadrant 3 judgments the autonomous architecture cannot make. The shape applies equally to the batch sub-form (reviewers attached to every agent verdict) and to the conversational sub-form (experts fact-checking each turn of a legal-consultation or diagnostic-support chat). (iv) The *accountability-chain dissolution at postmortem*: when the autonomous loop blends classification, routing, retrieval, and tool selection into one runtime stream, post-incident reconstruction has no architectural place to route consequence to; what presents as an unanswered redirect question is in fact a redirect question the architecture itself has foreclosed. The four symptoms are siblings; resolving the root — re-architecting Quadrant 3 work into its proper sub-form by I/O modality — resolves all four.

The distinction matters because it changes what counts as a fix. For the artificial case, sandboxing the autonomous loop does not fix anything: the architecture is wrong, and the right move is to move the work. For the principled case, sandboxing also does not fix anything, but for a different reason: the impossibility is not in the blast radius but in the redirect graph. What the principled case needs is not a stronger sandbox but a pre-named bearer of the gap that the architecture has already chosen to incur.

Figure 3a (optional) — Redirect paths under three configurations: success (redirect succeeds; accountability distributes), artificial impossibility (redirect dissolves; re-architecture is the fix), principled impossibility (redirect dissolves by design; pre-named gap-bearer is the structural commitment).

The conflation has a sustaining cause, named in the next section.

---

4. The LLM Workflow Quadrant Vocabulary Gap

The current industry vocabulary has no positive name for Quadrant 3.

The phrasing matters. "Anything that isn't deterministic" is not a name; it is a residual category. *AI agent*, *agentic workflow*, *agentic AI* — the dominant industry terms — denote something more like Quadrant 4 (the autonomous loop) than something orthogonal to it. The five workflow patterns Anthropic (2024) documents — *prompt chaining*, *routing*, *parallelization*, *orchestrator-workers*, *evaluator-optimizer* — together with OpenAI (2025)'s *manager pattern* and *decentralized pattern*, describe orchestration shapes that operate

in this territory, but the documents themselves do not name the frame; Anthropic titles its patterns *workflows* and contrasts them with *agents*, leaving the *workflow* category as the absence of *agent* rather than as a positively named cell. Anthropic's own warning — *"this might mean not building agentic systems at all"* (Anthropic 2024) — takes the same negative form as the broader cautionary discourse around agentic AI: do not build an agent when you do not need one. None of these warnings provides the positive name the practitioner on the ground needs in order to know what to build instead.

The vocabulary gap has an antecedent. In the RPA era, business automation tooling carved off Quadrant 1 work (deterministic, definable) and left the remaining semantic-judgment exception handling — what is now nameable as Quadrant 3 — on the operations floor as manual work. The era had no vocabulary for Quadrant 3 either, and the work that should have been factored out as bounded LLM functions or specialized chat agents was instead absorbed as human overhead. When LLMs arrived, the vocabulary did not arrive with them; the same Quadrant 3 work now gets handed off, by elimination again, to autonomous loops. The current accountability discourse is in part a repetition of the RPA-era pattern in agent vocabulary.

The consequence of the missing positive name is *routing by elimination*. When a team confronts a piece of work that involves semantic judgment, the choice presents itself as: *automate it deterministically*, or *give it to an agent*. The first option is Quadrant 1 or Quadrant 2; the second is read as Quadrant 4. Quadrant 3 — semantic judgment with a known path, bounded LLM calls with named roles and explicit schemas — does not appear in the choice list. Work that structurally belongs in Quadrant 3 gets routed to Quadrant 4 not because Quadrant 4 fits, but because Quadrant 3 is unnamed in the reasoning the team can perform.

The artificial redirect impossibility of Section 3 is the *industrial-scale reproduction* of this routing-by-elimination mechanism. Each individual choice — *give this to an agent* — is locally rational under the current vocabulary; the cumulative effect is that work which would tolerate post-hoc separation is implemented in architectures that destroy it. The mechanism is not a misunderstanding of any individual concept; it is the structural absence of a positive name for the cell that most current LLM applications actually inhabit.

The vocabulary gap is therefore *load-bearing*. Naming Quadrant 3 — explicitly, positively, alongside the other three — is the load-bearing vocabulary contribution of the framework. The name proposed here, *LLM Workflow Quadrant*, is intentionally austere: it names the architecture (workflow with LLM calls inside named roles) without inheriting connotations from the broader *agentic AI* discourse. Other names could carry the same work; the structural commitment is that the cell is named at all.

A consequence of naming is the *deflation* of the central place currently occupied by autonomous loops in the deployment landscape. If the LLM Workflow Quadrant is the positively named default for most LLM applications, then the Autonomous Agentic Loop Quadrant returns to the role for which it was originally designed: open-ended exploration where the path cannot be enumerated in advance. The shift is not a prohibition on autonomy; it is a recovery of the design space the autonomy was crowding out.

Naming the quadrant, however, does not finish the work. A further question remains: if a team correctly identifies Quadrant 3 work and implements it as a workflow at design time, does the workflow stay in Quadrant 3 once it is deployed and runs against patterns that did not appear in the design? The next section names the dimension along which this question must be answered.

---

5. Phase Separation: Design vs Operation

The framework so far is *quadrant-only*. Once Quadrant 3 is named and Quadrant 4 is restored to its proper scope, the diagnostic frame is half complete. The other half is a dimension that has been hiding in plain sight: the lifecycle phase a piece of work occupies at any given moment.

The *design phase* of a deployment is the phase in which a workflow's structure is being discovered or built. The execution path is not fully known; exploration is the requirement; the optimization axis is *flexibility*. Prototyping a workflow, running a research session, exploring an unbounded corpus, debugging a failing case — all of these are design-phase activities. The natural architecture for design-phase work is the one that maximizes the range of paths the agent can pursue: Quadrant 4 is at home here, but so is Quadrant 3 (rapid prototyping with bounded calls), Quadrant 2 (algorithm exploration), and Quadrant 1 (script-level scaffolding).

The *operation phase* of a deployment is the phase in which a workflow runs on its known path. The execution path is fixed; predictability is the requirement; the optimization axis is *predictability* in a broad sense — audit traceability, attribution, cost stability, conformance to a service-level commitment. The natural architecture for operation-phase work is the one that minimizes the variance between design-time expectation and runtime behavior: Quadrants 1 and 3 sit naturally here, with Quadrant 2 entering when explicit search is required. Quadrant 4 is *admissible* in the operation phase but carries an additional cost the next section makes explicit.

The two optimization axes are *inverted*. Flexibility and predictability are not orthogonal preferences; they pull against each other. A system that maximizes flexibility — that allows the next step to depend on the runtime situation — pays for that flexibility in audit-traceability, attribution, and variance from design-time intent. A system that maximizes predictability — that fixes the path at design time and runs it on every invocation — pays for that predictability in its inability to handle inputs the design did not anticipate. Each is a real trade; neither is a bug. The category error is to compress both into the same runtime and claim that capability improvements will make the trade-off disappear.

Phase and Quadrant are *independent* dimensions. The 4 × 2 cross-product is a *landscape*, not a mapping:

	Design phase	Operation phase
Script Quadrant	Prototype scripts	Production pipelines
Algorithmic Search Quadrant	Algorithm prototyping	Production solvers
LLM Workflow Quadrant	Prototype workflows	Production-default LLM workflows
Autonomous Agentic Loop Quadrant	Coding agents, deep research, exploratory R&D	Continuously-evolving research operations, exploratory analytics

Figure 3 — The Phase × Quadrant landscape. Each cell is admissible; the landscape is two-dimensional, not a partition. The framework's load-bearing observation concerns one cell (operation-phase Quadrant 4) and a default about the rest of the operation column.

The same workflow can appear in both phases of the same project: a routing prompt is prototyped in design and promoted to operation; an autonomous loop is used in design to explore the path of a problem before its findings are encoded into a workflow that goes into operation. The lifecycle is fluid in the appropriate direction: design-phase explorations *crystallize* into operation-phase workflows as their paths become known. The opposite direction — operation-phase work that requires runtime exploration — is the cell that needs explicit handling, addressed next.

The phase axis also reframes Sections 3 and 4. The surface diagnosis (misapplication) and the middle diagnosis (vocabulary gap) are *symptoms* of a deeper structural framing: the assumption that "an always-on autonomous agent runs the business" is a coherent target. Once the two phases are separated, the target is recognizable as a category compression: it asks design-phase flexibility and operation-phase predictability of the same runtime, and the only honest answer is that one of them loses.

---

6. The Phase-Crossing Decision

The load-bearing rule of the framework can now be stated.

Before placing a Quadrant 4 component as the operation-phase default of a deployment, the deploying team records, in one sentence, the *Phase-crossing decision*: when a new pattern surfaces in operation, will the autonomous loop handle it dynamically in place, or will it be routed back to the design phase as feedback?

The decision has exactly two admissible answers. *In place* means that the operation-phase runtime absorbs the new pattern dynamically, accepting the recurring attribution gap as the price of doing so. *Routed back to design* means that the operation surface returns to a default composition (Quadrants 1 and 3, with Quadrant 2 where applicable) until the design phase decides where the new pattern fits in the Quadrants and re-deploys. Both answers are admissible. "We will figure it out when it happens" is not admissible: it is the absence of a Phase-crossing decision masquerading as an answer.

The Phase-crossing decision is *additive*. It is required *in addition to* the design-time triage of Section 3 (which quadrant does this work belong in?) and *in addition to* the pre-named gap-bearer commitment of Section 3 (who is the organizationally identifiable human who absorbs the principled attribution gap?). The companion repository formalizes the gap-bearer's terminus as the rule that each agent process is bound, at deployment time, to exactly one identifiable human who is the *accountable operator* — approver of behavior-modifying changes, owner of incident response, endpoint of the accountability chain (Shimomoto 2026, ADR-0008). The Phase-crossing decision presupposes this chain: it is the deployment-time commitment of that named individual to a specific posture toward runtime emergence.

The rule is *procedural*, not architectural. It does not forbid Quadrant 4 in operation; it requires that the team's choice between in-place absorption and feedback-to-design be explicit at deployment time, recorded in the deployment artifact, and inspectable in post-incident reconstruction. A post-incident question that the rule answers is no longer "what did the agent do?" alone, but also "what Phase-crossing decision was the deployment operating under when this happened?" The answer is in the deployment record, not inferred.

An example, anonymized from the companion repository's case studies. Consider a contract-review system that combines clause classification (Quadrant 3, batch sub-form), jurisdiction inference (Quadrant 3, batch sub-form), and novel-clause analysis (Quadrant 4, since novel clauses are by definition not enumerable in advance). The system is admissible under the framework: the boundary between the workflow portion and the autonomous-loop portion is a structural choice, the autonomous-loop portion has a pre-named gap-bearer, and the workflow portion preserves post-hoc separability for its slice of the work. The Phase-crossing decision the team records at deployment time reads, in one sentence: *novel clauses surfaced in production are routed back to design as feedback; the next deployment of the workflow incorporates them as named cases.* The alternative — *novel clauses are handled in place by the autonomous-loop module* — is also admissible, but it commits the deployment to the recurring attribution gap of in-place absorption, and that commitment is what the gap-bearer is named for. Either answer makes the deployment accountable for the cost it has named.

A footnote on the operation-phase default. Most current autonomous-loop products are marketed as operation-phase deployments — the agent runs continuously, ingests new patterns, and adapts. Framework-aligned deployments need to make the Phase-crossing decision explicit even when the product materials assume "operation = autonomous-loop runtime" by default; the assumption is not wrong, but it is not the only honest framing, and the framing on which the deployment will be answerable in a postmortem is the one written into the deployment record.

Figure 4 — The Phase-crossing decision tree. Branch 1: in-place dynamic absorption (recurring attribution gap; pre-named gap-bearer carries it). Branch 2: feedback-to-design (operation surface returns to default composition; design phase decides where the new pattern fits). Both branches are admissible; the absence of a recorded branch is the only non-admissible state.

The next section asks whether the rule recurses below the level of business-system placements.

---

7. Descent to Skill-Design Gradient

The Phase axis applies recursively. Stated above at *business-system granularity* — the placement of Quadrants in a deployment — it descends to *skill-design granularity* and further to *skill-internal subcomponents*, where it explains several patterns that would otherwise look like exceptions. The load-bearing Phase-crossing decision continues to apply only at the business-system placement of Quadrant 4; the descent below it is informational, but informational in a way that reverses the dominant capability-axis reading of the Quadrant 3 ↔ Quadrant 4 boundary.

The first observation: *the same job lands at different positions on the Quadrant 3 ↔ Quadrant 4 gradient depending on phase.* A job whose target artifacts and execution environment are not yet known at design time naturally lives as a runtime-judgment skill (closer to Quadrant 4); the same job, once its targets settle into fixed paths and naming conventions in operation, can be re-implemented as a frozen pipeline of bounded calls (closer to Quadrant 3 or even Quadrant 1). The transition is a maturation, not a regression: the work has not been simplified; the path through it has been discovered. A skill that was correctly autonomous in design becomes correctly workflow-shaped in operation, with no loss of capability and a substantial gain in separability.

The second observation: *two secondary forces decide where on the gradient a skill (or subcomponent) lands.* The first is *target identifiability* — whether the artifacts the skill operates on sit at fixed paths or under fixed naming conventions (an invoice-processing skill in operation knows the incoming PDFs land in `s3://invoices/inbox/`; a research-assistant skill in design does not yet know which corpora the next session will need). High target identifiability is a precondition for operation-phase frozen-pipeline implementations: when targets are fixed, judgment that previously had to be made at runtime can be encoded as embeddings, clustering, threshold logic, or other deterministic machinery. The second is *scale-resilience* — how many units the skill processes per invocation cycle and how much miss rate is tolerable at that scale (an invoice-matching skill running at 10,000 units per day tolerates a radically smaller miss rate per call than the same skill running at 30 per day, even when the per-call task is identical). LLM judgment accumulates miss rate as unit count grows, so larger scales push the implementation toward deterministic shapes regardless of which phase the work is in. Both forces must hold for an operation-phase frozen-pipeline form to be honest; either failing pushes the work back across the gradient toward Quadrant 4.

The third observation: *capability is downstream of phase, not the primary lever.* The temptation in current discourse is to read every operation-phase placement of Quadrant 4 as a *capability-limited* solution that better models will eventually retire. The temptation is structurally wrong. Higher LLM capability does not

change target identifiability — the artifacts either sit at fixed paths or they do not. Higher LLM capability does not change scale-resilience — accumulating miss rate at scale is a property of the architecture, not of the per-call accuracy. Capability improvements move the work along the gradient marginally, but the primary lever — what determines whether a skill is in Quadrant 3 or Quadrant 4 — is the phase the skill is being implemented for. The capability axis is real; it is not the load-bearing axis.

The third observation has a corollary. It is tempting to read the gradient as capability-governed — to argue that as model capability improves, more work can move from Quadrant 3 to Quadrant 4 because "the model can handle the judgment." The framework reverses the direction. As phase clarifies — as the path through a piece of work becomes known — *more work moves from Quadrant 4 to Quadrant 3*, because the structural conditions for post-hoc separability come into view. Capability buys flexibility on the design side and reliability per call; phase governs where on the gradient the work belongs. Each is real; the order between them matters.

The descent has practical consequences for skill libraries and agent-harness design, which the companion repository's case studies and anti-patterns navigators operationalize (Shimomoto 2026, `quadrants/` directory). The framework's contribution at this granularity is the recognition that *the same diagnostic frame applies recursively*. When an autonomous-loop module is proposed for a skill, walk the three layers in order: misapplication first, vocabulary gap second, phase conflation third. Most resolutions surface at one of the first two layers before the third is even reached.

---

8. Relationship to Existing Frameworks

The framework above is *complementary* to existing AI risk-management and management-system standards, not a substitute for them. The major standards in the current landscape — the national AI risk-management framework structured as GOVERN / MAP / MEASURE / MANAGE functions (NIST 2023), and the international AI management-system standard structured around Plan / Do / Check / Act with an Annex A of organizational controls (ISO/IEC 2023) — provide *structural shape*: where the governance work happens, what cadence it runs on, what artifacts it produces. The framework here provides the *judgment layer* those structures presuppose: which decisions need to be made before the controls become applicable, and which architectural commitments determine whether the controls can succeed at all.

A specific consequence: the Phase axis is *orthogonal* to the Plan / Do / Check / Act cycle of the management-system standard. Plan / Do / Check / Act describes a control loop that runs *within each phase* — design has its plan/do/check/act loop (over the next prototype, the next experiment), and operation has its plan/do/check/act loop (over the next deployment cycle, the next metric drift). The Phase axis distinguishes which *optimization axis* the loop is on (flexibility-maximizing or predictability-maximizing). A management-system reader who reads the Phase axis as a refinement of Plan / Do / Check / Act will collapse the two distinctions: every Plan / Do / Check / Act variant lives inside one phase or the other, and the axis the framework names is the one *between* them. Two or three sentences in the standard's mapping document (Shimomoto 2026, `policy-mapping/`) make this explicit.

The relationship to AI risk-management functions is similar in shape. The GOVERN function records who decides; MAP characterizes the system in context; MEASURE produces evidence; MANAGE acts on the evidence. The framework's Quadrants and Phase axis are *inputs* to MAP and *commitments* the GOVERN function ratifies. The Phase-crossing decision is a GOVERN artifact (a recorded commitment); the gap-bearer naming is a GOVERN artifact; the quadrant-routing of each component informs the MAP function (system characterization in context). None of this is in conflict with the standard; the framework writes down what the standard leaves for each implementing organization to fill in.

The mapping should be expected to *decay* on the standards' revision cycles, not on the AAP framework's own. The repository's mapping documents track the standard versions explicitly and warn the reader that the mappings expire on the standards side; the architectural decisions on the AAP side remain stable across standard revisions, because they are framework-neutral by construction.

---

9. Conclusion and Open Questions

The framework records two structural commitments that the current AI-accountability discourse leaves implicit. The first is a *positive name* for the design space most current LLM applications inhabit, which deflates the residual-category routing that produces artificial redirect impossibility. The second is a *time-axis* — Phase Separation — that surfaces a Phase-crossing decision required at deployment time, distinct from but additive to the design-time quadrant triage and the pre-named gap-bearer commitment. Both commitments are *procedural*, not architectural. The framework does not forbid autonomous loops; it requires that the team's choice be named at deployment time, recorded in the deployment artifact, and inspectable after the fact.

Three observations close the paper.

Capability distribution is not accountability distribution. The dominant axis of current AI safety work is capability: what an agent can do, how strongly it can be sandboxed, how often it must defer to a human. The framework here is on a different axis: to whom what an agent has done can be redirected. Capability improvements do not produce accountability distribution. The accountability story has its own structural prerequisites — quadrant routing, phase separation, the Phase-crossing decision, the pre-named gap-bearer — and capability improvements are *downstream* of those structural prerequisites, not their source.

The framework is experimental. The two load-bearing rules (Sections 3 and 6) are stated in the companion repository as experimental architecture decision records (Shimomoto 2026, ADR-0009, ADR-0010). They have been discovered through implementation and operation of a single agent, not deduced from a governance theory. The open questions are the research agenda.

Implementation dissolves; judgment persists. The specific mechanisms by which the commitments are enforced — hooks, pipelines, command queues, audit records, JSONL logs — are not durable. They will be replaced. What persists is the judgment encoded in the commitments themselves: that Quadrant 4 carries a non-removable attribution gap, that operation-phase Quadrant 4 placements carry a second-order risk the design-time commitment does not cover, that the chain of accountability must terminate at a named individual. These are not implementations. They are decisions about who can be redirected to when the system fails.

Open question 1. Operationally measurable proxies for *post-hoc separability* in semantic-judgment work. The current criterion ("each call's contribution is separable") is a design judgment; empirical signals — call-graph stability across runs, role consistency under perturbation, schema compliance rates — may make the judgment more transferable across organizations, but the right metric set is open.

Open question 2. Operation-phase edge cases where the Phase-crossing decision "handle in place" is the structurally honest answer. High-frequency exploratory analytics, continuously-evolving research operations, real-time content moderation against unbounded patterns — the framework currently has no empirical signals for distinguishing "Phase-crossing in place is right here" from "this team is rationalizing a misapplication."

Open question 3. *What structural criterion reliably distinguishes a sequence of design sessions from an operation-phase autonomous-loop deployment, particularly in product-built systems where the same surface*

is used by both modes? Coding agents, deep-research tools, and exploratory analysts are continuously available but each session completes; their continuous availability is a deployment-level fact, not a phase-level property of each session. Non-accumulation of state across sessions in operation-relevant ways is a candidate criterion; whether it is sufficient — and whether it can be made operationally inspectable rather than left as a design-intent claim — is open.

Open question 4. Interaction with regulatory regimes. Some domains regulate the Autonomous Agentic Loop Quadrant directly. Whether the framework should reference such regimes by name in the triage decision, or remain architecturally framed and let regulation map onto the quadrants externally, is unresolved.

The framework names what was implicit. It does not close the questions; it makes them stable enough to ask.

References

Anthropic (2024). *Building Effective Agents*. https://www.anthropic.com/research/building-effective-agents

Elish, M. C. (2019). Moral Crumple Zones: Cautionary Tales in Human-Robot Interaction. *Engaging Science, Technology, and Society* 5: 40–60. https://doi.org/10.17351/ests2019.260

ISO/IEC (2023). *ISO/IEC 42001:2023 Information technology — Artificial intelligence — Management system.* International Organization for Standardization.

NIST (2023). *AI Risk Management Framework (AI RMF 1.0).* NIST.AI.100-1. National Institute of Standards and Technology. https://doi.org/10.6028/NIST.AI.100-1

OpenAI (2025). *A Practical Guide to Building Agents.* https://cdn.openai.com/business-guides-and-resources/a-practical-guide-to-building-agents.pdf

Shimomoto, T. (2026). *Agent Attribution Practice (AAP), v0.3.0.* Public repository and Architecture Decision Records. Zenodo. https://doi.org/10.5281/zenodo.20251893. Referenced ADRs: ADR-0008 (One Agent, One Human), ADR-0009 (Triage Before Autonomy), ADR-0010 (Phase Separation). Referenced supporting documents: `glossary.md`, `thesis.md`, `quadrants/`, `policy-mapping/`, `inspiration.md` (seven-essay narrative spine, Articles 1–7).

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2022). ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629.