SALESFUSION

Company Memory Is Not a Folder

Technical architecture and implementation report

PUBLIC RELEASE · VERSION 1.0.0 · 2026-07-17

Company Memory Is Not a Folder

Publication metadata

Field	Value
Author	SalesFusion Research
Publication date	2026-07-17
Version	1.0.0
Canonical slug	company-memory-is-not-a-folder
Canonical path	/technical-papers/company-memory-is-not-a-folder
Publication class	Technical architecture and implementation report
Approval state	Founder-approved public release

Revenue Cortex connection

Revenue Cortex depends on Company Memory because growth work cannot improve from stale folders or unsupported summaries. Buyer intelligence, positioning, outbound decisions, sales-call preparation, and pipeline review all require source-backed memory with provenance, freshness, conflict handling, and authority boundaries.

Evidence-status box

> This paper describes an operational pattern SalesFusion uses and proposes a broader Company Memory model. The pattern separates source-backed assertions, provenance, state, correction, and review from ordinary document storage. The broader model still requires benchmark evaluation for retrieval quality, conflict handling, staleness exclusion, and correction success. The allowed claim is architectural and methodological: useful organizational memory requires provenance, authority, freshness, conflict, uncertainty, and supersession semantics in addition to document retrieval. This paper presents no measured effect on revenue, decision quality, or user productivity.

Abstract

Organizational memory is often treated as a folder of documents plus search. That model fails when an agent or operator must answer operational questions under uncertainty: Which source is authoritative? When was the statement true? Is a newer source a correction, a conflict, or a supersession? Is the fact restricted? Is the

answer inferred, asserted, disputed, stale, or unknown? We define Company Memory as a source-governed memory architecture for business AI systems. It stores sources, assertions, entities, relationships, versions, epistemic state, provenance, restrictions, conflicts, corrections, and effective time. Search and vector indexes are treated as projections, not sources of truth. Obsidian or a similar note workspace is a readable projection and correction interface, not the canonical runtime. The model supports retrieval that returns evidence and state, not just text. It also defines failure modes and evaluation tests that must be passed before generated analysis can be trusted as an input to governed action.

The problem

A folder answers one question: where is the file? A business agent needs different questions answered. What does the company currently believe about an account, process, constraint, policy, metric, or workflow? Which statements are source-backed? Which statement was true during a prior time window? Which statement is restricted from public output? Which source has higher authority? Which assertions disagree? Which answer should be withheld because the evidence is stale or unresolved?

Document retrieval alone cannot represent these distinctions. A current policy, a deprecated draft, a generated summary, and an operator note may all rank highly for the same query. Without state semantics, the system can launder weak evidence into confident prose. Without effective time, it can use an old truth as a current fact. Without supersession, it can show a newer statement but leave the prior one ambiguously alive. Without restrictions, it can route sensitive material into an inappropriate output. Without correction records, it can hide the path by which the memory changed.

For consequential workflows, these failures matter. A recommendation based on stale authority should be blocked. A proposed action based on a disputed assertion should be escalated. A generated inference should remain an inference until accepted through the correct policy. The memory layer must therefore preserve the state of knowledge, not merely the text that produced it.

Contributions

This paper makes five contributions.

First, it defines Company Memory as a structured, event-backed memory layer with explicit epistemic states.

Second, it specifies provenance, effective time, restriction, correction, conflict, supersession, and retrieval semantics.

Third, it separates the canonical runtime from search and Obsidian projections.

Fourth, it gives concrete data objects, state transitions, invariants, and decision rules for memory operations.

Fifth, it defines an evaluation program for memory quality that does not require unsupported business-outcome claims.

Core definitions

Source is retained evidence with owner, tenant, classification, effective time, review state, version, and restrictions.

Assertion is a structured statement about a subject, predicate, and value. It references one or more sources and carries epistemic state.

Epistemic state is the memory's status label for a statement. The model separates epistemic status from lifecycle state. Epistemic status distinguishes observed_fact, authorized_assertion, inference, hypothesis, estimate, and unknown. Lifecycle state distinguishes active, stale, disputed, superseded, and retracted; a proposed record remains pending until validation and authority checks pass.

Effective time is the time interval during which a source or assertion is claimed to apply. It is separate from record creation time.

Supersession means a newer assertion replaces the current operational use of a prior assertion while preserving the prior assertion for audit and historical reconstruction.

Correction is a new event and usually a new assertion version. It does not erase the old record.

Restriction is a rule attached to a source, assertion, or field that limits retrieval, output, use, export, or audience.

Projection is a derived view such as search, vector index, report, dashboard, or Obsidian note. Projections can be rebuilt from canonical state.

Memory model and state protocol

The canonical memory has four record families: sources, assertions, entities and relationships, and events. A minimal assertion record is:

```
{
  "assertion_id": "asrt_active_constraint_001",
  "tenant_id": "tenant_demo",
  "subject_ref": "entity_revenue_motion",
  "predicate": "has_current_constraint",
  "value": "operator approval required for external actions",
  "epistemic_status": "authorized_assertion",
  "lifecycle_state": "active",
  "source_refs": ["src_authority_matrix_v1"],
  "authority_rank": "approved_source",
  "effective_from": "2026-07-01",
  "effective_to": null,
  "review_due": "2026-08-01",
  "restriction_refs": [],
  "supersedes": [],
  "conflict_set_ref": null,
  "version": 3
}
```

The assertion state machine is:

```
proposed -> active -> stale -> superseded
             |-> disputed -> active | superseded | retracted
             |-> retracted
```

Generated analysis may create a proposed assertion or an inferred assertion, depending on policy. It cannot silently create an active company fact. Activation requires source sufficiency, authority, restriction checks, and review state. A corrected source creates a new event and may create a new assertion version. A retracted source marks dependent assertions as stale, disputed, or retracted according to policy; it does not delete history.

Conflict is first-class. Two assertions can share subject and predicate while disagreeing on value, effective time, source authority, or restriction. The memory opens a conflict set rather than choosing last-write-wins. Retrieval must surface required unresolved conflicts when the objective depends on the predicate.

Supersession is also first-class. If a newer operating policy replaces an older one, the older assertion becomes superseded for current-state retrieval but remains available for historical questions. The system can reconstruct what it believed on a prior date by replaying effective time and event time.

Provenance, restrictions, and retrieval

W3C PROV-O provides useful vocabulary for entities, activities, agents, derivation, and attribution. The Company Memory model uses the same general provenance discipline: an assertion should be traceable to a source and to the activity that produced or changed it. The citation supports provenance vocabulary, not any claim that this implementation produces better outcomes.

Retrieval is not "top k chunks." A memory retrieval request specifies objective, effective time, minimum freshness, allowed classifications, required predicates, tenant, actor, and output use. The response contains:

Field	Meaning
active assertions	source-backed statements eligible for the request
inferences	generated or derived statements labeled as such
conflicts	unresolved disagreements relevant to the request
unknowns	required predicates with insufficient evidence
stale exclusions	records excluded because freshness requirements failed
restricted exclusions	records withheld because use or audience is not allowed
provenance	source refs, version refs, event refs, and locators
digest	integrity summary of the snapshot

Decision rules:

```
if requested_tenant != resource.tenant:
    deny("TENANT_SCOPE_DENIED")
if source_or_assertion.is_restricted_for(output_use):
    exclude_and_report("RESTRICTED")
if required_predicate.has_unresolved_conflict:
    return_state("CONFLICT_UNRESOLVED")
if required_predicate.only_has_stale_assertion:
    return_state("SOURCE_INSUFFICIENT")
if generated_inference_requested_as_fact:
    return_state("UNSUPPORTED_ASSERTION")
```

These rules preserve the difference between absence, restriction, conflict, staleness, and inference. A generated answer may still be useful, but it must carry its memory state.

Obsidian projection

The Obsidian projection exists because operators need a readable, editable surface. It can show source notes, entity pages, assertion ledgers, conflict pages, decision packets, run summaries, and learning records. The projection can also collect human corrections.

The projection does not own canonical truth. A direct note edit creates an import proposal with a diff, author, timestamp, target records, and rationale. The import proposal then passes validation:

Check	Purpose
schema validity	prevent malformed canonical records
source link	prevent unsupported fact creation
version match	prevent overwriting newer state
authority check	prevent unauthorized correction
restriction check	prevent inappropriate disclosure
conflict handling	prevent silent erasure

If approved, the canonical store emits events and the projection is regenerated. This keeps Obsidian useful without letting a local edit bypass memory governance.

Implementation

SalesFusion has implemented this pattern at the operational-design level: source-backed records, canonical state, correction surfaces, and redundancy controls are separated from free-form notes. The broader Company Memory architecture can be implemented with a relational canonical store, an append-only event log, retained source artifacts, and disposable search projections.

Core endpoints include:

```
POST /v1/sources
POST /v1/sources/{source_id}/versions
POST /v1/assertions
POST /v1/assertions/{assertion_id}/correct
POST /v1/assertions/{assertion_id}/dispute
POST /v1/assertions/{assertion_id}/supersede
GET  /v1/state/stale
GET  /v1/state/conflicts
POST /v1/state/snapshot
POST /v1/projections/obsidian
POST /v1/projections/obsidian/import-proposals
```

Mutating requests require idempotency keys and optimistic version checks. A key reused with a different request body is rejected. A correction never deletes the old assertion; it writes a new event and changes retrieval eligibility. Search indexes and vector indexes are rebuilt from canonical records and may be discarded.

The most important implementation invariant is this: every current-state value must be explainable as a source-backed assertion, an accepted inference, an unknown, a conflict, or an excluded restricted/stale record. Anything else is not Company Memory; it is unsupported generated text.

Evaluation

Memory evaluation should be independent of business outcomes. A benchmark corpus can include one authoritative fact, two consistent sources, contradictory sources with different authority, stale and fresh sources, inference misrepresented as fact, unsupported generated assertion, retracted source, corrected source, restricted source requested for public output, and source text containing prompt injection.

Metrics include source precision, current-state accuracy, conflict recall, stale-record exclusion, unsupported-assertion rate, uncertainty calibration, correction success, provenance completeness, and restriction compliance. The system should also be tested for cross-tenant retrieval. In pre-installation safety tests, cross-tenant leakage should be zero.

Acceptance criteria:

Criterion	Required behavior
stale exclusion	stale assertion cannot satisfy a freshness requirement
conflict visibility	unresolved required conflict blocks or escalates
correction trace	prior record remains auditable
restriction enforcement	restricted material is excluded from disallowed output
inference label	generated inference is not returned as fact
source trace	every active assertion has source and event references

These tests can show architecture correctness. They cannot show that the memory improves business performance without a separate field evaluation.

Safety, risks, and failure modes

The main memory risks are memory poisoning, prompt injection from source content, silent staleness, conflict erasure, retrieval laundering, restriction leakage, false confidence, and projection bypass. OWASP's LLM application threat list is relevant background for prompt injection and unsafe tool behavior, while NIST AI RMF supports a risk-management framing. Neither reference is evidence of SalesFusion-specific safety.

Controls include treating source text as data, preserving provenance, requiring activation policy, using conflict sets, excluding stale records by default from current-state retrieval, labeling inference, restricting output by audience and use, auditing projection imports, and preserving append-only event history. Zero-trust reasoning from NIST SP 800-207 supports resource-scoped checks, especially tenant isolation and least privilege.

Failure should be explicit. If the system cannot answer, it should return unknown, source_insufficient, conflict_unresolved, or restricted, not invent an answer. This is a product requirement, not a writing preference.

Limitations

The model adds operational overhead. Teams must maintain source ownership, review cadence, authority ranking, restrictions, and correction workflows. Some organizations have poor source hygiene; the memory layer will expose that problem rather than solve it automatically. The model also does not eliminate judgment. A high-authority source can be wrong, a correction can be incomplete, and a conflict can require human interpretation.

This paper does not provide a benchmark result. It specifies the benchmark that should be run. It also does not claim that Obsidian is the best projection for every organization; the architectural requirement is a readable projection with proposal-based import, not a specific editor.

Practical implications

A team building business agents should begin with memory semantics before building broad action. The minimum viable memory is not a document upload. It is a source registry, assertion ledger, conflict set, staleness rule, restriction model, correction path, and state snapshot endpoint.

The operating implication is also direct: when a human corrects the system, the correction must become structured memory, not a chat message trapped in a transcript. When an agent uses memory, it must carry source state forward into the decision packet. When a note is edited, it must become a proposal until validated.

References used

This paper uses W3C PROV-O for provenance vocabulary, NIST AI Risk Management Framework for risk framing, NIST SP 800-207 for resource-scoped access reasoning, and OWASP Top 10 for Large Language

Model Applications for threat awareness. These references do not validate SalesFusion-specific retrieval or business outcomes.

Conclusion

Company Memory is not a folder because company knowledge is not just text. It is sourced, time-bound, restricted, disputed, corrected, superseded, and sometimes unknown. A business AI system that cannot represent those states will eventually confuse retrieval with truth. The SalesFusion pattern separates canonical state from human-readable projection and generated analysis. The broader model should now be tested with retrieval, conflict, staleness, correction, restriction, and tenant-isolation benchmarks before stronger claims are made.

Changelog

- 2026-07-17 — Version 1.0.0: founder-approved first public release; publication metadata, evidence-status box, limitations, and verified references retained.

SALESFUSION

The First Running Job

Technical architecture and implementation report

PUBLIC RELEASE · VERSION 1.0.0 · 2026-07-17

The First Running Job

Publication metadata

Field	Value
Author	SalesFusion Research
Publication date	2026-07-17
Version	1.0.0
Canonical slug	first-running-job
Canonical path	/technical-papers/first-running-job
Publication class	Technical architecture and implementation report
Approval state	Founder-approved public release

Revenue Cortex connection

A Revenue Cortex becomes credible when one real revenue job starts running with evidence before broader automation is attempted. The First Running Job is the activation wedge: a bounded, inspectable workflow such as a wake-up list, stale-deal radar, context packer, follow-up router, CRM truth loop, or approval console.

Evidence-status box

> This paper describes an activation protocol SalesFusion uses and proposes an evaluation design. The protocol uses an Activation Card, non-mutating proof, evidence contract, approval boundary, live gate, rollback path, and expansion criteria. The allowed claim is protocol-level: a non-mutating, evidence-producing first job provides a more inspectable activation unit than launching broad autonomous scope at once. Any safety advantage remains a hypothesis until comparative activation and incident evidence is collected.

Abstract

The first live workflow in an agentic business system sets the operating pattern for everything that follows. If the first deployment is broad, mutating, weakly evidenced, and vaguely approved, later governance becomes cleanup. We define the First Running Job as a narrow activation unit that tests whether a workflow can use source-backed state, respect authority, produce expected artifacts, record run evidence, and pass a live gate before expansion. This paper specializes the broader Revenue OS installation sequence defined in Paper 06.

The protocol centers on an Activation Card: objective, scope, sources, authority, simulation plan, evidence contract, approval boundary, live criteria, rollback plan, and expansion criteria. The first job should run in simulation or non-mutating proof mode before any external consequence. Completion requires artifact and receipt verification, not process exit or agent self-report. SalesFusion uses the method as an activation protocol, and it still requires comparative evaluation before claims about lower incident rates or faster activation.

The problem

Agentic systems often fail at activation because the first workflow is chosen for excitement rather than inspectability. A broad workflow touches many sources, decisions, and external systems. When it fails, the team cannot tell whether the issue was memory quality, authority, connector permission, prompt behavior, evidence contract, or operational review. If the first workflow mutates external systems, errors can create cleanup work before governance is understood.

The more inspectable engineering move is to make the first job narrow and evidence-producing. The goal is not to establish that the whole system is valuable. The goal is to test the activation path: memory is source-backed, authority is explicit, simulation is possible, expected artifacts are defined, live approval is scoped, completion is verified, and rollback is known.

Contributions

This paper contributes the First Running Job protocol.

First, it defines the Activation Card as the pre-run contract for objective, scope, evidence, approval, live gate, rollback, and expansion.

Second, it specifies simulation and non-mutating proof as the default path before live activation.

Third, it defines an evidence contract for completion that rejects process exit or agent self-report as sufficient proof.

Fourth, it separates approval boundary, execution boundary, and expansion boundary.

Fifth, it proposes evaluation criteria for comparing activation approaches without claiming results.

Core definitions

First Running Job is the first narrow workflow activated in a governed business AI system. It is selected to test the operating path, not to maximize scope.

Activation Card is the structured artifact that must be complete before the job runs.

Simulation is a run against fixtures, dry-run adapters, or read-only data where no external mutation occurs.

Non-mutating proof is a real or realistic run that produces an artifact, recommendation, report, or verification output without changing an external business system.

Evidence contract is the predefined set of artifacts, receipts, logs, and checks required for verified_complete.

Approval boundary is the exact action, version, scope, actor, approver role, and expiry covered by approval.

Live gate is the decision point that allows the job to move from proof to bounded live operation.

Expansion criteria are the conditions required before adding more scope, actions, sources, or autonomy.

Activation Card

The Activation Card is the core object. It forces vague launch decisions into explicit fields by requiring the team to write the evidence contract before the run.

```
{
  "activation_card_id": "act_first_job_001",
  "job_name": "read_only_state_snapshot_job",
  "objective_refs": ["obj_activation_reliability"],
  "scope": {
    "tenant": "tenant_demo",
    "included_sources": ["src_current_policy_v1"],
    "excluded_actions": ["external_mutation"],
    "population": "bounded_fixture_or_approved_segment"
  },
  "memory_requirements": ["required_assertions", "freshness", "known_conflicts"],
  "authority": {
    "consequence_class": "read_only_or_internal_reversible",
    "approval_required_for_live": true,
    "approval_scope": "exact_workflow_version_and_action"
  },
  "simulation_plan": "run_with_non_mutating_adapter_and_expected_fixture",
  "evidence_contract": ["artifact_schema", "run_events", "trace", "blocker_state",
"verification_check"],
  "live_gate": ["proof_passed", "owner_reviewed", "rollback_tested"],
  "rollback_plan": "return_to_prior_version_or_disable_job",
  "expansion_criteria": ["zero_unauthorized_actions", "verified_completion", "operator_can_pause"]
}
```

The card should be short enough to review and precise enough to test. If a field is unknown, the job is not ready.

Protocol and state transitions

The First Running Job protocol has nine steps:

1. Select a narrow job with known objective, bounded scope, and known failure cases.
2. Build the Company Memory snapshot needed for the job.
3. Define the consequence class and authority boundary.
4. Write the evidence contract and expected artifact schema.
5. Run simulation or non-mutating proof.
6. Verify artifacts, events, blockers, traces, and restrictions.
7. Review the Activation Card and proof output.

8. Pass or fail the live gate.
9. If approved, run bounded live operation and monitor expansion criteria.

The run state machine is:

```
requested -> started -> output_created -> verification -> verified_complete
                |-> blocked -> resumed
                |-> failed -> retry | cancelled | rollback
```

The activation state machine is:

```
draft_card -> proof_ready -> proof_running -> proof_review
            -> live_gate_passed -> bounded_live -> expansion_review
            -> blocked | rejected | rollback
```

A blocked proof is a useful result. It can reveal missing sources, stale memory, wrong authority, connector drift, prompt injection, duplicate action risk, or an unrealistic artifact contract.

Evidence contract

The evidence contract defines what counts as completion. It should include:

Evidence	Purpose
input snapshot digest	identifies which memory state was used
workflow version	provides a check against approval/version mismatch
run events	reconstructs execution path
trace id	links infrastructure observation
expected artifact	shows useful output was produced
artifact validator	checks schema or content requirements
blocker record	explains incomplete or paused work
action receipt	records external action evidence, if any
verification result	separates output from verified completion
operator review state	records human decision

For a non-mutating proof, the action receipt may be a dry-run receipt or explicit no-mutation receipt. For a later live action, the receipt must be tied to the exact connector action and idempotency key.

Completion rule:

```
if process_exit == success and expected_artifact_missing:
    status = "verification_failed"
if artifact_valid and required_receipts_present and restrictions_respected:
    status = "verified_complete"
else:
    status = "blocked_or_failed"
```

This rule provides a deterministic check against false-completion claims. It also gives the team a concrete basis for debugging.

Approval boundary and live gate

Approval must be scoped to the exact workflow version and consequence class. A proof run may require only read-only authority. A live run that mutates an external system requires a separate approval. Approval includes action digest, version, scope, approver role, expiry, and conditions. Approval does not execute the action.

The live gate should require:

Gate item	Required evidence
memory	required assertions active, fresh, and sourced
conflicts	no unresolved required conflict
proof	non-mutating proof completed or blocked with accepted resolution
authority	approval valid for exact action and version
rollback	rollback or disable path tested
operator control	pause, inspect, reject, and export available
reporting	run evidence appears in reporting ledger

If any item fails, the correct output is not "try anyway." It is block, collect evidence, correct memory, revise the workflow, or escalate.

Rollback and expansion criteria

Rollback is part of activation, not an emergency add-on. For the first job, rollback may mean disabling the workflow, reverting to a prior workflow version, retracting a proposed memory change, or reconciling an external action if the live job was allowed to mutate. The rollback plan should be tested before broad scope.

Expansion criteria define what must be true before adding more workflows, broader populations, or more autonomy:

Criterion	Example
verified completion	proof or live runs meet evidence contract
no unauthorized action	authority suite records zero violations
no cross-tenant leakage	isolation tests pass
operator comprehension	operator can explain sources and approval path
blocker handling	known blockers are classified and reviewed
rollback proof	disable or version rollback works
reporting visibility	outcomes, costs, approvals, and failures visible

Expansion should be denied if the first job has unreviewed failures, unresolved authority ambiguity, missing rollback, or evidence gaps.

Implementation

SalesFusion uses the First Running Job as an activation protocol within the broader installation design. The implementation uses Company Memory, authority checks, workflow simulation, run evidence, reporting, and operator review.

Relevant interfaces:

```
POST /v1/state/snapshot
POST /v1/policies/evaluate
POST /v1/workflows/{workflow_id}/simulate
POST /v1/workflows/{workflow_id}/runs
POST /v1/runs/{run_id}/verify
POST /v1/runs/{run_id}/rollback
POST /v1/approvals
GET  /v1/runs/{run_id}
GET  /v1/audit/actions/{action_id}
```

Implementation invariants:

Invariant	Failure prevented
no broad first scope	unclear failure diagnosis
no mutation before proof	premature external consequence
no approval reuse	wrong-version execution
no self-reported completion	false done state
no hidden expansion	gradual authority creep
no untested rollback	unrecoverable activation

OpenTelemetry can provide trace and metric vocabulary; domain events provide business proof. RFC 9110 can support consistent HTTP interface semantics. Neither reference validates activation outcomes.

Evaluation

A proper evaluation would compare activation approaches. Possible conditions include broad immediate launch, narrow live launch, and First Running Job with non-mutating proof. Measures could include incident rate, unauthorized-action rate, false-completion rate, time to diagnose failure, rollback success, operator comprehension, and expansion decision quality. The study would need comparable settings, baseline, sample, time window, and disclosed limitations.

Before any field comparison, a scenario test suite should cover missing source, stale assertion, unresolved conflict, expired approval, wrong workflow version, prompt injection in source material, connector failure, duplicate retry, missing expected artifact, rollback unavailable, and attempted scope expansion.

Acceptance criteria for the protocol:

Criterion	Required behavior
incomplete card	job cannot start
proof failure	live gate blocks
artifact missing	run cannot be verified complete
approval mismatch	execution denied
duplicate retry	prior receipt returned or duplicate blocked
rollback unavailable	high-consequence live gate escalates
expansion request	checked against criteria before approval

These tests support protocol correctness. They do not prove safety advantage without comparative evidence.

Safety, risks, and failure modes

The First Running Job can fail if the job is narrow but irrelevant, non-mutating but unrealistic, approved by the wrong role, verified against a weak artifact, or expanded despite unresolved issues. Another risk is theater: teams may create an Activation Card but ignore it when launch pressure appears.

Controls include required card completeness, explicit exclusion lists, proof run evidence, authority binding, live gate review, rollback test, and expansion criteria. NIST AI RMF supports risk framing; OWASP LLM application threats are relevant to prompt injection and unsafe tool use; NIST SP 800-207 supports resource-scoped access reasoning. These references do not validate the protocol's incident effects.

Limitations

A First Running Job is not a full-system proof. It tests a narrow path. It may miss failures that appear only under larger volume, more connectors, more ambiguous objectives, or more consequential actions. A non-mutating proof can also understate live risk because real external systems behave differently from dry-run adapters.

The protocol also requires human discipline. If reviewers approve vague cards, skip rollback, or expand scope for convenience, the structure loses force. The method is best treated as a gate with audit consequences, not a checklist for appearance.

Practical implications

The first job should be selected for diagnosis value. A good candidate has bounded sources, clear expected output, known failure cases, non-mutating proof path, and visible objective relevance. A poor candidate requires broad authority, many connectors, unclear completion, or irreversible external consequences.

Operators should expect the first job to teach the installation what is missing. A blocked proof is not a failed project; it is evidence that the system found a boundary before live mutation. Expansion should follow verified operation, not the mere absence of visible errors.

References used

This paper uses NIST AI Risk Management Framework for risk framing, NIST SP 800-207 for resource-scoped access reasoning, OWASP Top 10 for Large Language Model Applications for threat awareness, OpenTelemetry Specification for observability concepts, and RFC 9110 for HTTP interface semantics. These references do not validate SalesFusion-specific activation outcomes.

Conclusion

The First Running Job is the smallest useful activation unit for a governed business AI system. It makes the launch question concrete: can this system use source-backed state, respect authority, produce evidence, pass review, and recover from failure on one bounded workflow? SalesFusion uses this activation protocol, but

comparative safety and incident claims still require evaluation. The practical rule is simple: test one narrow job with evidence before expanding autonomy.

Changelog

- 2026-07-17 — Version 1.0.0: founder-approved first public release; publication metadata, evidence-status box, limitations, and verified references retained.

SALESFUSION

Ask Once, Analyze Many

Technical architecture and implementation report

PUBLIC RELEASE · VERSION 1.0.0 · 2026-07-17

Ask Once, Analyze Many

Publication metadata

Field	Value
Author	SalesFusion Research
Publication date	2026-07-17
Version	1.0.0
Canonical slug	ask-once-analyze-many
Canonical path	/technical-papers/ask-once-analyze-many
Publication class	Technical architecture and implementation report
Approval state	Founder-approved public release

Revenue Cortex connection

Revenue Cortex should not ask a founder to repeat the same company facts for every asset, agent, audit, or campaign. A canonical-state model lets approved inputs about the product, buyer, offer, objections, proof, and constraints feed many analyses while keeping each downstream use reviewable and bounded.

Evidence-status box

> This paper describes an implemented SalesFusion interaction-architecture redesign and states its evidence limits. SalesFusion removed 24 duplicate designed tasks across the first eight modules of a Revenue OS design artifact and corrected the portal interaction model around canonical state and delta-only review. That count is a design artifact count, not a measured client outcome, time-savings result, or usability result. The broader claim remains to be evaluated: canonical state and delta-only review are intended to remove repeated client input while preserving distinct analytical and authority boundaries.

Abstract

Multi-stage agentic workflows often ask humans the same question repeatedly because each module treats its own form, prompt, or checklist as the source of truth. Repetition creates fatigue, inconsistency, and review noise, but removing all review is unsafe. We define "Ask Once, Analyze Many" as an interaction architecture in which human input becomes canonical state, later stages request only deltas or unresolved fields, and analysis remains separate from authority. A client or operator answers a question once when the answer is stable,

source-backed, and properly scoped. Analytical agents can reuse that canonical state for multiple tasks, but they cannot convert analysis into approval or mutate consequential systems without the required authority. The paper specifies canonical state objects, delta review, boundary rules, state transitions, evaluation design, and failure modes. It is an implementation report about interaction design, not evidence of client productivity improvement.

The problem

Repeated questioning is common in consulting portals, onboarding workflows, RevOps audits, and AI-assisted operating systems. A module asks for objectives. A later module asks for the same objective in a different phrasing. A third workflow asks for the target segment again because it does not trust the prior answer. The person answering begins to copy, abbreviate, or contradict prior responses. The system then has more text but less reliable state.

The opposite failure is also common. A system asks once, stores a vague answer, and reuses it everywhere without checking whether the later use has a different audience, authority requirement, effective time, or restriction. That creates hidden overreach. Input reuse is only safe when the system knows what the answer means, where it came from, when it applies, and what it may authorize.

The required architecture is therefore not "ask fewer questions." It is canonical state with deltas, evidence, review states, and authority boundaries.

Contributions

This paper contributes four implementation concepts.

First, it defines canonical state as the reusable record created from human input, source evidence, accepted corrections, and versioned review.

Second, it defines delta-only review: later stages present what changed, what is missing, what conflicts, and what requires authority, rather than asking for everything again.

Third, it separates analytical reuse from approval. Reusing an answer for analysis does not authorize an external action or policy change.

Fourth, it states an evidence-limited implementation result: 24 duplicate designed tasks were retired from a design artifact. The paper does not claim measured effort reduction.

Core definitions

Canonical state is the structured, versioned record that later workflows read. It contains values, sources, owners, effective time, review status, restrictions, and history.

Question instance is a human-facing request for information. It maps to one or more canonical fields.

Delta is a change, missing field, conflict, stale field, restriction issue, or authority issue relative to the last accepted canonical state.

Delta-only review is a review mode in which the user sees only the fields requiring attention and the reason each field requires attention.

Analytical boundary is the limit between using state to generate analysis and treating that analysis as accepted company fact.

Authority boundary is the limit between analysis or recommendation and consequential action.

Review state is the status of a canonical field: draft, proposed, active, stale, disputed, superseded, restricted, or unknown.

Interaction architecture

The architecture begins with a field map. Each question must declare which canonical field it affects, what source or actor can answer it, whether the value is stable or time-bound, and what later workflows may use it for.

```
{
  "field_id": "company.current_priority_constraint",
  "question_text": "What constraint currently limits the operating motion?",
  "required_source": "owner_review_or_source_record",
  "effective_time_required": true,
  "allowed_uses": ["analysis", "internal_recommendation"],
  "disallowed_uses_without_approval": ["external_action", "policy_change"],
  "review_cadence": "monthly"
}
```

The canonical field record:

```
{
  "field_id": "company.current_priority_constraint",
  "value": "operator approval required for external actions",
  "state": "active",
  "source_refs": ["src_authority_matrix_v1"],
  "owner_role": "business_owner",
  "effective_from": "2026-07-01",
  "effective_to": null,
  "last_reviewed_at": "2026-07-15",
  "review_due_at": "2026-08-15",
  "restriction_refs": [],
  "version": 4
}
```

Later modules do not ask the original question again unless a rule requires it. They request a state snapshot. The response can say:

State	User-facing behavior
active and fresh	do not ask again; show in review context if needed
missing	ask once and bind to canonical field
stale	ask for confirmation or new source
disputed	show conflict and request resolution
restricted	hide or request authorized reviewer
changed	show delta and ask for acceptance
analysis-only	allow reuse for reasoning, not action

This creates a practical difference between repeated input and repeated confirmation. The system should not re-collect stable facts. It should still surface meaningful deltas and approvals.

Analytical and authority boundaries

The design separates four states that are often collapsed:

Boundary	Allowed movement	Not allowed
input to canonical state	answer becomes proposed or active field after validation	raw text silently becomes universal truth
canonical state to analysis	agents reuse source-backed fields for analysis	agents ignore restrictions or freshness
analysis to recommendation	system proposes options with evidence	recommendation becomes approval
approval to execution	approved action can be executed by separate step	approval event directly mutates external system

This matters because asking once can otherwise become unsafe. A user may answer a strategy question for internal analysis. That answer should not automatically authorize external contact, commercial commitment, publication, or production mutation. Consequence class and policy still apply.

The architecture also guards against generated analysis contaminating canonical state. An agent may infer a likely missing field, but the inferred value remains proposed or inferred until accepted through the right review path.

Delta-only review protocol

A delta review packet contains:

Field	Meaning
changed fields	accepted value, proposed value, source, and diff
missing fields	required fields absent for this workflow
stale fields	fields past review date or effective interval
conflicts	contradictory active or proposed values
restricted fields	values unavailable for current actor or output
authority requests	approvals needed for the next consequential step
no-change digest	integrity summary of reused state

The review algorithm:

```
snapshot = canonical_state.snapshot(objective, workflow_version, actor)
delta = compare(snapshot, workflow_requirements)

if delta.has_missing_required_fields:
    ask_questions(delta.missing)
if delta.has_stale_fields:
    request_confirmation_or_source(delta.stale)
if delta.has_conflicts:
    request_resolution(delta.conflicts)
if delta.has_authority_requests:
    create_approval_packet(delta.authority)
if delta.is_empty:
    proceed_to_analysis_with_digest(snapshot.digest)
```

The user should be able to inspect reused state without being forced to retype it. The system should also be able to explain why a question is being asked again. Acceptable reasons include staleness, conflict, changed use, changed authority, restriction, or missing source.

Implementation

SalesFusion implemented a redesign that retired 24 duplicate designed tasks across the first eight modules of a Revenue OS artifact. The redesign changed the interaction model from repeated module-local questions to canonical state and delta review. The count is only evidence that designed duplication was removed from the artifact. It does not prove that users saved time, made fewer errors, or completed work faster.

An implementation requires:

Component	Role
canonical field registry	maps questions to reusable state
source and assertion ledger	preserves provenance and effective time
workflow requirement schema	declares needed fields and authority
delta engine	computes missing, changed, stale, disputed, restricted fields
review UI	presents deltas and accepts corrections
approval service	handles consequential authority separately
audit events	records question, answer, reuse, correction, approval

Events include field.proposed, field.activated, field.corrected, field.marked_stale, field.superseded, review.delta_created, review.accepted, review.rejected, approval.requested, and decision.recommended.

The design uses the Company Memory principle that correction creates a new record or version; it does not erase prior state. Obsidian or a portal can be the human surface, but canonical state lives in structured records.

Evaluation

The correct evaluation is not an artifact count alone. A before/after study should measure human task success, time on task, repeated-question exposure, correction rate, contradiction rate, decision quality judged by reviewers, authority comprehension, and support requests. It should also record null and negative results.

Scenario tests should include stable fields reused across modules, fields that become stale, conflicting answers from two reviewers, a restricted answer requested for public output, an answer valid for analysis but not external action, and a generated inference attempting to become canonical state.

Acceptance criteria:

Criterion	Required behavior
no duplicate collection	active fresh field is not re-asked as blank input
explain repeat	repeated question has a visible reason
delta accuracy	changed, missing, stale, disputed, restricted fields are correctly labeled
boundary preservation	analysis reuse does not create approval
correction integrity	prior value remains auditable
authority separation	consequential action still requires policy decision

This evaluation would support usability and interaction claims only if method, sample, baseline, and results are published together.

Safety, risks, and failure modes

The main risk is over-reuse. A field that was correct for one workflow may be wrong for another because the effective time, audience, authority, or restriction differs. Another risk is under-review: the system hides a reused field that should have been confirmed. The opposite risk is false duplication, where a system asks again because it cannot map question variants to the same canonical field.

Controls include field-level effective time, allowed-use metadata, restriction checks, review cadence, authority classification, delta reasons, and audit events. NIST AI RMF supports the general need to manage AI-system risk; W3C PROV-O supports provenance vocabulary for state and source relationships. These references do not establish usability outcomes.

Limitations

Canonical state requires careful field design. If fields are too broad, reuse becomes unsafe. If fields are too narrow, users face unnecessary deltas. Delta-only review can also hide context if the interface does not let reviewers inspect the full state when needed.

The 24-task correction is an implementation artifact count. It is useful evidence that the design was changed, but it is not evidence that client effort decreased. A proper study may find smaller benefits, no benefits, or tradeoffs between reduced repetition and increased need to understand canonical state.

Practical implications

Teams building multi-stage agent workflows should create a canonical field registry before expanding modules. Every repeated question should be classified as one of four cases: same canonical field, genuinely new field, stale confirmation, or authority-specific review. This classification turns "ask less" from a preference into a rule.

Operators should monitor repeated-question reports and correction logs. If users repeatedly correct a reused value, the field definition, review cadence, or source authority may be wrong. If users repeatedly approve actions without reading deltas, the authority UI needs redesign.

References used

This paper uses W3C PROV-O for provenance vocabulary and NIST AI Risk Management Framework for risk framing. These references do not validate SalesFusion-specific usability or business outcomes.

Conclusion

Ask Once, Analyze Many is a canonical-state architecture, not a slogan. The system should collect stable information once, preserve its evidence and limits, and reuse it across analytical workflows through delta-only

review. SalesFusion implemented a design correction that retired 24 duplicate tasks from an artifact, but the practical effects remain unevaluated. The next step is a before/after usability and decision-quality study with visible method and limitations.

Changelog

- 2026-07-17 — Version 1.0.0: founder-approved first public release; publication metadata, evidence-status box, limitations, and verified references retained.

SALESFUSION

Governed Outbound Readiness as a State Machine

Technical architecture and implementation report

PUBLIC RELEASE · VERSION 1.0.0 · 2026-07-17

Governed Outbound Readiness as a State Machine

Publication metadata

Field	Value
Author	SalesFusion Research
Publication date	2026-07-17
Version	1.0.0
Canonical slug	governed-outbound-readiness-state-machine
Canonical path	/technical-papers/governed-outbound-readiness-state-machine
Publication class	Technical architecture and implementation report
Approval state	Founder-approved public release

Revenue Cortex connection

Revenue Cortex should not treat outbound as ready because a message sounds persuasive. In the revenue-specific system, outbound readiness is a compound state across fit, source evidence, identity, buyer route, claims, suppression, copy review, expiry, and human approval before any send or launch.

Evidence-status box

> SalesFusion uses explicit outbound readiness states as a workflow control pattern. The pattern distinguishes fit, signal evidence, contact verification, route confidence, claim risk, suppression, quality review, approval, expiry, and invalidation. This paper defines the state machine and implementation expectations. It does not claim improved response rates, reduced compliance risk, higher conversion, or better deliverability. Those claims require a false-ready, suppression, and policy-violation benchmark and, separately, a field outcome evaluation.

> The design uses risk-management framing consistent with the NIST AI Risk Management Framework, provenance concepts consistent with W3C PROV-O, and LLM-application threat awareness consistent with the OWASP Top 10 for Large Language Model Applications.

Abstract

Outbound readiness is often implemented as a copy-completion flag. If a message exists, the record is marked ready. That model lacks the controls required for governed AI-assisted outbound work. A message can be grammatically complete while the account fit is weak, the signal is stale, the contact is unverified, the route is speculative, the claim risk is high, suppression applies, approval has expired, or new information has invalidated the premise. Governed readiness is a compound state machine.

This paper defines an outbound readiness model in which each dimension has its own state, transitions, blockers, expiry, and invalidation rules. It is the action-gating specialization of the buyer-intelligence pipeline described in Paper 09. The final readiness state is derived from the dimensions; it is not edited directly by a model or operator. The architecture is intended for systems that prepare external messages or recommendations and must block or flag unsupported, stale, suppressed, or unauthorized action. SalesFusion uses this as a pattern; safety and operational effects remain evaluation questions.

The problem

Outbound workflows combine several risks. They use business claims about a target organization, identity claims about a contact, channel and suppression rules, human approvals, and generated language. A single error can create external harm: contacting the wrong person, citing an unsupported fact, using copied source language, ignoring suppression, sending after approval expiry, or acting after a source has changed.

The usual implementation pattern is too coarse. A CRM task may say research complete. A copy generator may say draft ready. A reviewer may say approved. A scheduler may say queued. None of those alone means safe to send. Research can be complete but low confidence. Copy can be polished but unsupported. Approval can be valid for an earlier version. A queued action can become invalid when suppression changes or evidence expires.

The practical answer is not more free-form caution text. It is a machine-readable readiness state with explicit dimensions, transitions, and denial reasons.

Contributions

This paper contributes:

- A compound readiness state machine for outbound action.
- A dimension table covering fit, signal evidence, contact verification, route confidence, claim risk, suppression, QA, approval, expiry, and invalidation.
- Decision rules for deriving not_ready, ready_for_review, approved_pending_execution, ready_to_execute, expired, and invalidated.
- Implementation invariants that block a draft, approval, or queue position from becoming readiness evidence by rule.
- An evaluation design focused on false-ready records, suppression failures, stale evidence, and policy violations.

Definitions

Outbound action means any externally visible message, contact attempt, publication, profile update, CRM mutation, or other external consequential step related to outreach.

Readiness dimension is one independent component of readiness. It has a state, evidence, reviewer or system owner, expiry, and invalidation rules.

Derived readiness is the final state computed from dimensions. It is not manually set.

False-ready means the system marks an outbound action executable even though at least one required dimension is blocking, expired, invalidated, or unknown.

Suppression means a rule, list, policy, preference, legal basis, channel constraint, or operational exclusion that blocks or limits action.

Claim risk is the risk that outbound copy asserts more than the evidence supports, uses restricted source language, implies private knowledge, or makes an unapproved representation.

Compound readiness model

The readiness object has one record per proposed outbound action. The action is versioned and digest-bound. Changing the target, channel, message, evidence, workflow version, or claim set changes the digest and forces re-evaluation.

Dimension	Allowed states	Blocking states
Fit	fit_confirmed, fit_plausible, fit_unknown, fit_rejected	fit_unknown, fit_rejected when fit is required
Signal evidence	current_supported, weak_supported, stale, disputed, missing	stale, disputed, missing
Contact verification	verified, role_possible, unverified, mismatch	unverified, mismatch
Route confidence	high, medium, low, not_applicable	low when route is required
Claim risk	low, moderate, high, prohibited	high without review, prohibited
Suppression	clear, limited, matched, unknown	matched, unknown where policy requires certainty
QA	not_started, passed, failed, needs_changes	not_started, failed, needs_changes
Approval	not_required, requested, approved, rejected, expired, revoked, mismatched	requested, rejected, expired, revoked, mismatched
Expiry	valid, expiring_soon, expired	expired
Invalidation	none, evidence_changed, target_changed, policy_changed, message_changed	any change requiring re-evaluation

The derived states are:

Derived state	Meaning
not_ready	One or more mandatory dimensions are missing, blocking, or unknown.
ready_for_review	Evidence and identity are sufficient for human or policy review, but approval or QA is incomplete.
approved_pending_execution	Approval exists but execution gate has not run final checks.
ready_to_execute	All mandatory dimensions pass at execution time for the exact action digest.
blocked	A dimension fails and requires evidence, correction, or authority.
expired	Time-bound readiness or approval has passed its validity window.
invalidated	A material change requires the state to be rebuilt.
executed	An action attempt occurred and has a receipt or blocker record.

Readiness is computed late. A record can be approved_pending_execution and still fail the final gate because suppression changed, evidence expired, or the contact identity was invalidated.

Transition rules

The state machine starts with a proposed action packet:

```
{
  "readiness_id": "rd_001",
  "action_digest": "sha256:...",
  "target_ref": "account_or_person",
  "channel": "external_message",
  "message_version": "msg_v3",
  "claim_refs": ["claim_1", "claim_2"],
  "workflow_version": "wf_v5",
  "derived_state": "not_ready"
}
```

Each dimension transitions through explicit events:

```
fit.assessed
signal_evidence.verified
contact_identity.verified
route_confidence.scored
claim_risk.reviewed
suppression.checked
qa.completed
approval.decided
readiness.expired
readiness.invalidated
readiness.recomputed
```

The derivation function is deterministic:

```
function derive_readiness(record):
  if record.invalidation != none:
    return invalidated
  if record.expiry == expired:
    return expired
  if suppression in [matched, unknown_when_required]:
    return blocked
  if signal_evidence in [missing, stale, disputed]:
    return blocked
  if contact_verification in [unverified, mismatch]:
    return blocked
  if claim_risk == prohibited:
    return blocked
  if qa in [failed, needs_changes]:
    return blocked
  if approval in [rejected, revoked, mismatched, expired]:
    return blocked
  if qa != passed or approval == requested:
    return ready_for_review
  if approval == approved and final_execution_checks_not_run:
    return approved_pending_execution
  if all required dimensions pass:
    return ready_to_execute
```

This function deliberately treats unknowns as blockers where the consequence class requires certainty. Unknown suppression, unknown contact identity, and missing evidence do not become low-confidence readiness. They become blocked or not_ready.

Implementation

SalesFusion uses this as an explicit state model around outbound preparation. The implementation pattern has three key properties.

First, readiness is derived. Operators and models can update dimension evidence through controlled events, but they cannot directly set ready_to_execute. This blocks a generated draft or checklist note from bypassing the state machine by rule.

Second, readiness is bound to an action digest. The digest includes target, channel, message version, workflow version, evidence references, claim set, suppression check timestamp, QA result, and approval reference. If the copy changes, if the route changes, if the contact changes, or if required evidence changes, the prior readiness becomes invalidated or mismatched.

Third, the execution adapter reruns final checks. Queue time can be long enough for evidence, approval, or suppression state to change. The adapter evaluates the current readiness record immediately before any external action and records the policy decision, idempotency key, and receipt.

The interface can be implemented with endpoints such as:

```
POST /v1/outbound-actions
POST /v1/outbound-actions/{id}/evidence
POST /v1/outbound-actions/{id}/suppression-check
POST /v1/outbound-actions/{id}/qa
POST /v1/outbound-actions/{id}/approval
POST /v1/outbound-actions/{id}/derive-readiness
POST /v1/outbound-actions/{id}/execute
```

The public interface should expose blocker reasons in structured form. A reviewer should see contact_identity=mismatch or claim_risk=prohibited, not just a red icon.

Evaluation

The near-term evaluation is a state-machine benchmark. It should use fixed scenarios with expected readiness states and expected blockers. Scenarios should include stale signal evidence, copied source language, unverified contact, wrong organization, low route confidence, suppression match, unknown suppression, high claim risk, QA failure, approval for a prior message version, expired approval, and evidence change after queueing.

Metrics:

Metric	Definition
False-ready rate	Proportion of blocked scenarios incorrectly marked executable.
False-block rate	Proportion of executable scenarios incorrectly blocked.
Suppression miss rate	Suppressed or unknown-suppression records marked clear.
Approval mismatch detection	Changed action versions detected before execution.
Expiry enforcement	Expired evidence or approval blocks action.
Invalidation latency	Time between material change and readiness invalidation.
Reviewer traceability	Reviewer can identify the blocking dimension and evidence.

Field evaluation should come later and should not be conflated with this benchmark. Deliverability, reply rate, or revenue outcomes require separate design, baseline, sample, time window, and confounder recording.

Safety and failure modes

The most serious failure is a false-ready external action. Common drivers include treating copy completion as readiness, relying on an old suppression check, reusing approval after message edits, sending to an unverified contact, allowing a model to soften high claim risk, or failing to invalidate queued work when evidence changes.

Another failure is silent overblocking. A system can become unusable if every uncertain but low-consequence action is blocked without clear reason or route to resolution. The remedy is not to weaken consequential gates; it is to define consequence classes and allow lower-risk internal actions, such as drafting or research, under different readiness rules.

Prompt injection in source material is also a risk. Source text should never instruct the outbound workflow. It is evidence input only. Copy generation must use approved claim records, not raw source text.

Limitations

This state machine does not decide whether outbound work is strategically worthwhile. It does not produce better copy by itself. It cannot ensure that a verified contact will welcome contact or that a signal reflects actual need. It also depends on the quality of suppression data, source review, and human approval.

The current evidence supports saying that SalesFusion uses explicit readiness states. It does not support comparative claims about safety, efficiency, conversion, or compliance outcomes.

Practical implications

Teams implementing AI-assisted outbound should replace ready with a derived state. Product surfaces should show dimension states, blocker reasons, expiry, and invalidation history. A draft should be treated as one artifact in the workflow, not proof that action is allowed. Approval should be tied to the exact version that will be executed. Suppression should be checked before execution, not only when research begins.

This model also changes review behavior. Reviewers can approve a claim but reject a message, approve a message but require a fresh suppression check, or mark a route plausible while keeping action blocked until identity is verified.

Conclusion

Outbound readiness is a compound evidence state, not a copy-completion flag. Governed execution requires fit, current evidence, verified identity, route confidence, acceptable claim risk, suppression clearance, QA, valid approval, unexpired state, and no material invalidation. SalesFusion uses this readiness-state pattern and proposes a benchmark to measure false-ready, suppression, expiry, and policy failures. Until such evaluation is reported, the proper claim is architectural and operational: explicit readiness states make the grounds for action inspectable.

References

- W3C, "PROV-O: The PROV Ontology." https://www.w3.org/TR/prov-o/
- National Institute of Standards and Technology, "AI Risk Management Framework." https://www.nist.gov/itl/ai-risk-management-framework
- OWASP Foundation, "OWASP Top 10 for Large Language Model Applications." https://owasp.org/www-project-top-10-for-large-language-model-applications/

Changelog

- 2026-07-17 — Version 1.0.0: founder-approved first public release; publication metadata, evidence-status box, limitations, and verified references retained.

SALESFUSION

Proof-of-Run for Recurring Agents

Technical architecture and implementation report

PUBLIC RELEASE · VERSION 1.0.0 · 2026-07-17

Proof-of-Run for Recurring Agents

Publication metadata

Field	Value
Author	SalesFusion Research
Publication date	2026-07-17
Version	1.0.0
Canonical slug	proof-of-run-for-recurring-agents
Canonical path	/technical-papers/proof-of-run-for-recurring-agents
Publication class	Technical architecture and implementation report
Approval state	Founder-approved public release

Revenue Cortex connection

Revenue Cortex needs proof that recurring revenue work actually ran, not just that an agent or cron job started. Wake-up lists, signal scans, pipeline reviews, follow-up routers, and reporting loops need run receipts, artifacts, no-result handling, and verification before they count as operating evidence.

Evidence-status box

> SalesFusion uses proof-of-run as an operating pattern for recurring agent work. The pattern records schedule events, starts, exits, artifacts, verification results, blockers, failures, no-result states, retries, reconciliation, and rollback. The design addresses the failure class in which a scheduled job or process exit does not establish useful completion. This paper does not report incident rates or claim comparative reliability improvement.

> The design uses observability concepts consistent with the OpenTelemetry Specification and append-only history concepts. It also follows the implementation principle that a successful HTTP or process response does not prove downstream completion.

Abstract

Recurring agents are easy to schedule and hard to prove. A cron trigger can fire, a worker can start, a process can exit with success, and an agent can report that it is done while the intended artifact is missing, invalid, stale, duplicated, or unusable. For business workflows, assignment and exit are not completion. Proof-of-run requires a state model that distinguishes scheduled, started, exited, output-created, verified-complete, blocked, failed, and no-result states.

This paper defines a reliability protocol for recurring agents. Paper 13 is its companion at the outcome layer: proof-of-run establishes execution evidence, while an outcome ledger records what happened after execution without asserting causation. The protocol specifies expected artifacts, idempotency keys, action receipts, traces, reconciliation checks, retry rules, blocker classification, and rollback behavior. It treats no_result as a valid terminal state when the agent ran correctly but no qualifying work existed. It treats blocked as distinct from failed when the run could not proceed because of missing evidence, authority, dependency, or external system state. SalesFusion uses this pattern and proposes an evaluation suite for false completion and incident-rate comparison.

The problem

Recurring agents create a special kind of ambiguity. They run on a schedule, not always in response to a human watching the work. If they fail silently, the system may assume ongoing coverage that does not exist. If they retry without idempotency, they may duplicate consequential work. If they report success without artifact verification, operators cannot distinguish "nothing needed to happen" from "the agent did not check" or "the check ran but produced unusable output."

The common status vocabulary is too small: pending, running, success, failed. It hides the difference between a schedule event and a worker start. It hides process exit from business completion. It hides the difference between no qualifying input and a broken query. It hides whether an external action was attempted once, twice, or not at all.

AI agents increase this risk because they can produce plausible narrative status. A message saying "completed" may be only a self-report. It is not an artifact, not a connector receipt, not a reconciliation result, and not a verified run state.

Contributions

This paper contributes:

- A proof-of-run state model for recurring agents.
- A required artifact contract for verifying completion.
- A distinction between blocked, failed, and no_result.
- Idempotency, reconciliation, retry, and rollback rules for recurring work.
- An implementation pattern using run records, event envelopes, traces, metrics, logs, artifacts, and receipts.

- An evaluation design for false completion, duplicate work, blocker quality, and recovery behavior.

Core definitions

Scheduled means a trigger or scheduler created an expected run. It does not prove a worker started.

Started means a worker accepted the run with a workflow version, input digest, idempotency key, and trace identifier.

Exited means the worker process returned. Exit can be successful, failed, timed out, or cancelled. It does not prove expected artifacts exist.

Output-created means the run produced an artifact or action receipt matching a known output type. It may still be invalid.

Verified-complete means the expected artifact contract passed deterministic checks and any required reconciliation.

Blocked means the run could not proceed because a dependency, authority, source, connector, policy, or human input was missing or invalid.

Failed means the run attempted work and encountered an error that prevented completion.

No-result means the run executed the required checks and correctly found no qualifying work or no qualifying output for the period.

Proof-of-run architecture

The protocol separates scheduling and orchestration events from evidence of useful work.

State	Evidence required	Typical next state
scheduled	Schedule event with expected run window	started, missed, cancelled
started	Worker claim with workflow version and trace ID	exited, blocked, failed
exited	Process exit code, duration, logs	output_created, failed, verification_failed
output_created	Artifact reference or action receipt	verified_complete, verification_failed
verified_complete	Artifact contract passed, reconciliation passed	terminal or outcome review
blocked	Blocker class and required resolution	resumed, cancelled, expired
failed	Error class, retry eligibility, trace	retry, rollback, cancelled
no_result	Query/check evidence proving no qualifying work	terminal

The minimal run record includes:

```
{
  "run_id": "run_001",
  "workflow_id": "wf_daily_review",
  "workflow_version": "wf_v12",
  "tenant_id": "tenant_ref",
  "scheduled_for": "2026-07-16T09:00:00Z",
  "input_digest": "sha256:...",
  "idempotency_key": "tenant:workflow:window",
  "trace_id": "trace_001",
  "expected_artifacts": ["review_packet", "run_summary"],
  "status": "started"
}
```

The artifact contract is specific to the workflow. A reporting run may require a dated report, source query digest, reconciliation totals, and validation result. A monitoring run may require a checked source list, qualifying event count, no-result proof when empty, and alert packet when non-empty. A mutating run may require an action receipt from the external system and an internal reconciliation event.

State transitions and decision rules

Proof-of-run is strict about transition meaning:

```
scheduled -> started -> exited -> output_created -> verified_complete
                    \-> blocked -> resumed -> exited
                    \-> failed -> retry -> started
                    \-> no_result
```

no_result should not be treated as failure. It is a successful negative result when the workflow's expected job is to check for qualifying work. But it must be evidenced. A blank output is not enough. The run must record the query or check performed, the source set, the time window, and the criteria that produced zero qualifying records.

Retries must be idempotent. A retry can re-run analysis, but a retry of a consequential external action must either return the prior receipt or prove no prior action occurred. A reused idempotency key with a different request body should be rejected.

Rollback depends on action class. Internal reversible actions can often be rolled back by restoring prior state or superseding a record. External consequential actions may not be fully reversible. The run record must distinguish rollback performed, rollback unavailable, and rollback requires human escalation.

Implementation

SalesFusion uses the proof-of-run pattern with run ledgers, expected-artifact checklists, blocker states, and reconciliation checks. The runtime captures schedule event, start event, process exit, trace identifier, logs, metrics, artifacts, action receipts, verification result, and outcome review as separate records. OpenTelemetry-compatible traces, metrics, and logs provide the observability vocabulary; the domain ledger provides the business state.

The implementation pattern uses these interfaces:

```
POST /workflows/{workflow_id}/runs
GET  /runs/{run_id}
POST /runs/{run_id}/heartbeat
POST /runs/{run_id}/artifact
POST /runs/{run_id}/block
POST /runs/{run_id}/fail
POST /runs/{run_id}/verify
POST /runs/{run_id}/retry
POST /runs/{run_id}/rollback
```

Verification is deterministic where possible. A run cannot mark itself verified by writing prose. The verifier checks artifact existence, schema validity, source window, required fields, action receipts, duplicate detection, and reconciliation rules. For recurring runs, the verifier also checks that the expected schedule window is covered and not double-counted.

Blockers are typed:

Blocker class	Example condition
source_unavailable	Required source could not be read.
authority_missing	Consequential action lacks valid approval.
evidence_insufficient	Required source or claim is missing, stale, or disputed.
connector_ambiguous	External system response cannot prove action state.
rate_limited	Connector or policy requires delay.
human_input_required	Workflow needs a decision or correction.

This classification matters because retrying an authority blocker is not the same as retrying a transient connector timeout.

Reconciliation, retries, and rollback

Reconciliation answers: did the world end up in the state the run believes it created? For internal artifacts, reconciliation can compare artifact digest, schema, and ledger entry. For external actions, it can compare connector receipts, external object identifiers, timestamps, and idempotency keys. For reporting, it can compare source totals and deterministic aggregations.

Retry rules:

```
if blocker is authority_missing:
  do not retry automatically
if error is transient_connector_failure:
  retry within capped policy using same idempotency key
if prior action receipt exists:
  return prior receipt and mark duplicate suppressed
if request body changed under same idempotency key:
  reject and audit
if repeated verification failure:
  stop retries and escalate
```

Rollback rules should be declared before execution. A workflow that cannot roll back a consequential action should require stricter approval. A rollback event should reference the original run, action receipt, rollback method, result, and remaining residual risk.

Evaluation

The immediate evaluation is a reliability benchmark, not a business outcome study. It should include scenarios where a schedule fires but the worker never starts, worker starts but exits before artifact creation, process exits successfully with invalid artifact, connector returns ambiguous response, retry would duplicate an action, no qualifying input exists, required authority is missing, and rollback is unavailable.

Metrics:

Metric	Definition
Verified-completion accuracy	Runs marked complete only when artifact contract passes.
False-completion rate	Runs marked complete despite missing or invalid evidence.
Missed-run detection	Scheduled but unstarted runs are detected.
No-result correctness	Empty periods are correctly distinguished from failures.
Duplicate-work rate	Retries do not duplicate consequential actions.
Blocker classification quality	Blockers are assigned actionable classes.
Retry recovery	Eligible transient failures recover without violating idempotency.
Rollback proof	Rollback records show what was reverted and what remains.

Incident-rate comparison requires a baseline system and a defined operating window. Until that exists, the proper claim is that the protocol is implemented and testable.

Risks and failure modes

False completion is the primary risk. It can come from trusting process exit, trusting agent self-report, accepting empty output, or skipping reconciliation. Duplicate consequential action is another serious risk, especially when

retries occur after timeouts or ambiguous connector responses. Silent missed runs are also dangerous because recurring work can appear to be covered while no worker is actually running.

Other risks include log loss, artifact tampering, cost runaway from retry loops, stale schedule windows, and rollback theater where a system records rollback without proving reversion. Controls include append-only events, integrity hashes where needed, idempotency, capped retries, dead-letter queues, independent verification, and human escalation for irreversible actions.

Limitations

Proof-of-run does not prove that the agent chose the best action. It only proves that a defined run occurred, produced or did not produce expected artifacts, and passed or failed verification. It depends on correct artifact contracts. A bad contract can verify the wrong thing. It also adds latency and storage overhead.

SalesFusion's current evidence supports use of an operating pattern that addresses the described failure classes. It does not establish their frequency and does not support quantified claims about reduced incidents, increased reliability, or improved business outcomes.

Practical implications

Recurring agents should be operated like auditable jobs, not like chat sessions. Each run needs a schedule record, start record, trace, artifact contract, verification result, blocker or failure class, and reconciliation result. Operators should review blocked and no-result states, not only failures.

Product teams should remove success as a standalone terminal state. Use verified_complete, no_result, blocked, failed, cancelled, and rolled_back instead. A process exit code can be useful telemetry, but it is not proof of business completion.

Conclusion

Proof-of-run is the evidence layer recurring agents need before their outputs can be treated as operationally verified. It separates scheduled, started, exited, output-created, verified-complete, blocked, failed, and no-result states; requires expected artifacts and reconciliation; and treats idempotency, retries, and rollback as first-class controls. SalesFusion uses this pattern and proposes reliability tests for false completion, duplicate work, missed runs, blocker classification, and rollback behavior.

References

- OpenTelemetry, "OpenTelemetry Specification." https://opentelemetry.io/docs/specs/otel/
- IETF, "RFC 9110: HTTP Semantics." https://www.rfc-editor.org/rfc/rfc9110.html

Changelog

- 2026-07-17 — Version 1.0.0: founder-approved first public release; publication metadata, evidence-status box, limitations, and verified references retained.

SALESFUSION

Outcome Ledgers for Business AI

Technical architecture and implementation report

PUBLIC RELEASE · VERSION 1.0.0 · 2026-07-17

Outcome Ledgers for Business AI

Publication metadata

Field	Value
Author	SalesFusion Research
Publication date	2026-07-17
Version	1.0.0
Canonical slug	outcome-ledgers-for-business-ai
Canonical path	/technical-papers/outcome-ledgers-for-business-ai
Publication class	Technical architecture and implementation report
Approval state	Founder-approved public release

Revenue Cortex connection

Revenue Cortex needs outcome ledgers because growth systems are otherwise tempted to attribute pipeline, meetings, replies, and revenue to whichever agent produced the most confident narrative. Deterministic ledgers keep source records, contribution categories, reconciliation, and causal interpretation separate.

Evidence-status box

> SalesFusion uses an internal outcome-ledger and deterministic validation model for business AI reporting. The model preserves source records, metric definitions, effective time, contribution categories, outcome events, reconciliation checks, and learning records. This paper defines the architecture and describes deterministic reconciliation as an internal operating pattern. It does not make revenue, attribution, forecasting, decision-quality, or organizational-performance claims. Cross-system reconciliation and proxy-gaming evaluation remain required.

> The design uses append-only event history concepts and observability concepts consistent with the OpenTelemetry Specification. Risk framing is consistent with the NIST AI Risk Management Framework.

Abstract

Business AI systems are increasingly asked to report outcomes: pipeline movement, task completion, experiment results, operational progress, and contribution to objectives. If those reports are generated directly from prose, dashboards, or model summaries, the system can silently alter totals, conflate proxies with outcomes, erase null results, and overstate contribution. Outcome reporting needs a deterministic ledger first and AI explanation second.

This paper defines an outcome-ledger architecture for business AI. It is the outcome-layer companion to Paper 11: proof-of-run records verified execution, while the ledger records subsequent observations and limits causal interpretation. The ledger records source records, deterministic metrics, contribution categories, effective time, outcome events, reconciliation results, proxy risks, and learning records. AI may explain the ledger, propose interpretations, identify anomalies, or draft review notes. It may not invent events, overwrite metric definitions, backdate outcomes, silently change totals, or promote correlation to causation. SalesFusion uses an internal ledger and validator pattern; broader field claims require cross-system and proxy-gaming evaluation.

The problem

Business reporting is already difficult without AI. Source systems disagree. Timestamps differ. Records are duplicated. A task can be completed without contributing to an objective. A local metric can move favorably while a downstream guardrail worsens. A campaign can appear successful because the population changed. A team can count created artifacts rather than useful outcomes.

AI adds another risk: fluent explanation can make a weak ledger look rigorous. A model can summarize incomplete data, fill missing reasons, smooth over reconciliation errors, or imply that a workflow produced an outcome because the workflow happened before the outcome. That is not measurement. It is narrative over ambiguous records.

The problem is not that AI should be absent from reporting. The problem is that AI must be bounded by deterministic records. It should explain what the ledger contains and does not contain. It should not become the ledger.

Contributions

This paper contributes:

- A deterministic outcome-ledger model for business AI systems.
- Definitions for source records, metrics, effective time, outcome events, contribution categories, reconciliation, proxy risks, and learning records.
- A boundary between AI explanation and metric computation.
- Decision rules for preserving null, blocked, failed, negative, and indeterminate outcomes.
- An implementation pattern with validators and append-only events.
- An evaluation design for cross-system reconciliation and proxy-gaming resistance.

Core definitions

Source record is an immutable or versioned reference to the system, event, artifact, or human review that supports an outcome entry.

Deterministic metric is a metric computed by declared rules from source records. Its definition includes numerator, denominator, inclusion rules, exclusion rules, time window, effective-time semantics, deduplication logic, and owner.

Effective time is the business time at which an event applies, distinct from the time the system recorded it.

Outcome event is a dated record that something relevant to an objective occurred, did not occur, failed, was blocked, or was found indeterminate.

Contribution category labels the relationship between a system action and an outcome. Examples include direct execution evidence, assisted human action, prerequisite activity, correlated event, guardrail breach, no observed contribution, and unknown.

Reconciliation is the process of comparing ledger records, source records, external system totals, and metric definitions to detect missing, duplicated, stale, or inconsistent data.

AI explanation boundary is the rule that AI may summarize, question, and explain ledger records but may not create metric totals or alter outcome state without a validated event.

Proxy risk is the risk that a measured proxy can move favorably while the intended business outcome does not, or while a guardrail worsens.

Learning record is a bounded conclusion from reviewed evidence. It includes confidence, applicability limits, counterevidence, and whether a change proposal is warranted.

Ledger architecture

The outcome ledger is an append-only event system with deterministic projections. It is not a dashboard assembled from mutable notes. Each metric view is reconstructed from source records and metric definitions.

Layer	Function	Example objects
Source layer	Captures evidence references	source record, artifact, receipt, review
Event layer	Records business and workflow events	outcome event, blocker, failure, guardrail breach
Metric layer	Defines deterministic computation	metric definition, observation, window
Reconciliation layer	Checks totals and consistency	reconciliation run, discrepancy
Explanation layer	Produces bounded narrative	AI summary, anomaly question, review note
Learning layer	Records reviewed conclusions	learning record, change proposal

A minimal outcome event:

```
{
  "outcome_event_id": "out_001",
  "tenant_id": "tenant_ref",
  "objective_ref": "obj_retention_example",
  "metric_ref": "metric_verified_completion",
  "event_type": "verified_completion",
  "effective_at": "2026-07-16T00:00:00Z",
  "recorded_at": "2026-07-16T01:00:00Z",
  "source_refs": ["run_001", "artifact_001"],
  "contribution_category": "direct_execution_evidence",
  "value": 1,
  "confidence_label": "deterministic",
  "limitations": ["does_not_measure_business_value"]
}
```

The limitation field is important. A verified completion event may prove that a workflow produced an artifact. It does not prove that the artifact improved revenue, saved time, or changed a decision.

Metric definitions and contribution categories

Metric definitions must be explicit enough to be tested. A metric record includes:

```
{
  "metric_id": "metric_false_completion_rate",
  "name": "False completion rate",
  "numerator_rule": "runs_marked_complete_without_valid_artifact",
  "denominator_rule": "runs_marked_complete",
  "time_window": "calendar_week",
  "effective_time_field": "run_completed_at",
  "deduplication_key": "run_id",
  "exclusions": ["cancelled_before_start"],
  "owner_role": "operations_reviewer",
  "version": 3
}
```

Contribution categories provide controls against overclaiming:

Category	Meaning
direct_execution_evidence	The system directly performed the counted action and has receipt or artifact proof.
assisted_human_action	The system contributed a draft, analysis, or packet, but a human performed the action.
prerequisite_activity	The work prepared conditions but is not the final outcome.
correlated_event	The outcome occurred near the work but contribution is not established.
guardrail_breach	A protected metric worsened or policy was violated.
no_observed_contribution	Review found no supported contribution.
unknown	Evidence is insufficient to classify contribution.

This vocabulary lets reports say less when less is known. It also makes the mistake of treating every workflow output as an outcome visible and blockable.

Reconciliation and AI explanation boundaries

Reconciliation should run before explanation. It checks schema validity, source existence, duplicate keys, window boundaries, effective-time handling, metric-version consistency, and cross-system totals where available. A report with unresolved reconciliation errors should be labeled incomplete.

AI explanation can be useful after reconciliation:

- summarize which metrics moved;
- identify which records drove a change;
- surface missing data or anomalies;
- draft questions for review;
- compare outcome events with guardrails;
- explain contribution categories and limitations.

AI explanation must not:

- invent missing source records;
- change metric totals;
- silently drop null, failed, blocked, or adverse events;
- infer causality from sequence;
- rewrite contribution categories;
- promote a proxy improvement into business outcome success;
- approve a workflow or policy change.

The model can propose a learning record, but the learning record must be reviewed and bounded.

Implementation

SalesFusion uses an internal outcome-ledger and validator pattern. The implementation stores metric definitions, source references, outcome events, reconciliation runs, discrepancies, AI explanations, learning records, and change proposals as separate records. Deterministic validators run before a report is treated as current.

The core flow is:

```
capture source record or artifact
record outcome event with effective time
validate schema and source references
deduplicate by metric-specific key
compute deterministic metric projection
run reconciliation checks
label unresolved discrepancies
generate bounded explanation
create learning record only after review
create change proposal if evidence supports it
```

A learning record includes evidence, counterevidence, confidence, applicability limits, metric version, window, and reviewer decision. It cannot directly modify a workflow, policy, or metric definition. Any modification requires a change proposal, approval, versioning, and rollback plan.

Evaluation

The first evaluation should test ledger correctness, not business impact. A benchmark can provide synthetic source systems with known duplicates, late-arriving events, changed metric definitions, missing records, guardrail breaches, and proxy traps. The system should compute the expected totals and label unresolved discrepancies.

Metrics:

Metric	Definition
Reconciliation accuracy	Correctly identifies matching, missing, duplicated, and inconsistent records.
Metric projection accuracy	Deterministic totals match expected results for the declared definition.
Effective-time accuracy	Events are counted in the correct business window.
Null preservation	No-result, blocked, failed, and adverse states remain in reports.
Explanation faithfulness	AI summaries match ledger records and limitations.
Proxy-risk detection	Local metric gains with guardrail breaches are flagged.
Causal-overreach rate	Reports avoid unsupported causal language.

Business-outcome evaluation requires additional design: baseline, comparison group or other credible design, measurement window, sample, concurrent changes, confounders, and preserved null or negative results.

Risks and failure modes

The main failure is narrative overreach. A report can imply that AI produced an outcome because the AI system produced an artifact earlier. Another failure is proxy gaming: optimizing for easy-to-count activities such as completed tasks, messages drafted, or meetings scheduled while the actual objective does not move in the intended direction or a guardrail worsens.

Metric corruption is also common. Wrong joins, duplicate records, stale source snapshots, changed definitions, and inconsistent effective time can distort totals. AI explanation can hide these errors if reconciliation status is not visible.

Controls include deterministic metric definitions, append-only events, source references, reconciliation runs, guardrail metrics, contribution categories, learning-record limits, and review gates for change proposals.

Limitations

Outcome ledgers do not solve causal inference. They make records cleaner and claims more bounded, but they do not prove that a workflow caused a business result. They also depend on source-system quality and correct metric definitions. If the organization chooses poor proxies, the ledger can faithfully report the wrong thing.

SalesFusion's current evidence supports an internal ledger and deterministic reconciliation model. It does not support claims about cross-system accuracy at scale, resistance to gaming in the field, or business performance improvement.

Practical implications

Teams should separate measurement from explanation. Compute metrics deterministically, then let AI explain the computed records with visible limits. Reports should include reconciliation status, metric version, effective window, source coverage, contribution categories, and guardrail state. Null, blocked, failed, and adverse outcomes should remain visible.

Learning should be cautious. A completed experiment can create a learning record. It should not automatically change workflow policy, metric definitions, claims, or permissions.

Conclusion

Business AI reporting needs outcome ledgers before persuasive explanations. Deterministic metrics, source records, effective time, outcome events, contribution categories, reconciliation, proxy-risk controls, and learning records define checks intended to keep AI from inventing or silently altering totals. SalesFusion uses this internal ledger and validation pattern. The next evidence step is cross-system reconciliation and proxy-gaming evaluation, not stronger claims about business outcomes.

References

- OpenTelemetry, "OpenTelemetry Specification." https://opentelemetry.io/docs/specs/otel/
- National Institute of Standards and Technology, "AI Risk Management Framework." https://www.nist.gov/itl/ai-risk-management-framework

Changelog

- 2026-07-17 — Version 1.0.0: founder-approved first public release; publication metadata, evidence-status box, limitations, and verified references retained.

SALESFUSION

A Reference Architecture for the Company Cortex

Technical architecture and implementation report

PUBLIC RELEASE · VERSION 1.0.0 · 2026-07-17

A Reference Architecture for the Company Cortex

Publication metadata

Field	Value
Author	SalesFusion Research
Publication date	2026-07-17
Version	1.0.0
Canonical slug	company-cortex-reference-architecture
Canonical path	/technical-papers/company-cortex-reference-architecture
Publication class	Technical architecture and implementation report
Approval state	Founder-approved public release

Revenue Cortex connection

In SalesFusion's commercial work, the Revenue Cortex is the revenue-specific application of this Company Cortex reference architecture. The Revenue Cortex narrows the same memory, attention, authority, execution, telemetry, and improvement layers around growth work: positioning, buyer intelligence, outbound, sales assets, pipeline operating cadence, and proof-of-run.

Evidence-status box

> SalesFusion has implemented and uses a V1 Company Cortex every day through an Obsidian-based operating environment. That V1 connects source-backed Company Memory, attention and priorities, decision and authority records, agent/workflow evidence, proof-of-run, and an operating cadence. This is operational evidence that the Cortex is an implemented working system rather than only an architectural proposal.

> The current V1 is assembled from Obsidian, structured operating records, existing tools, agents, and governed workflows; it is not presented as one proprietary integrated software runtime. Some services in the complete five-layer reference architecture remain proposed or only partially implemented, and the unified architecture has not yet been evaluated across a controlled client cohort. The strongest allowed claim is therefore implemented and operationally observed internally: SalesFusion has built a functional V1 and uses it daily, while the paper defines the more complete architecture that V1 is evolving toward. No causal business-outcome, revenue-effect, safety-improvement, or comparative-superiority claim is made without a separate evaluation.

Abstract

Most business AI deployments begin with a task agent, a retrieval index, or a workflow automation. Those components can be useful, but they do not by themselves answer the harder systems question: how should a company represent what it believes, what it wants, what it is allowed to do, what it actually did, and what it learned? We propose the Company Cortex as a reference architecture for that question. The design separates five layers: Company Memory, Executive Control, Telemetry, Execution, and an Improvement Controller. Human authority, tenant isolation, provenance, and exportability cut across all layers. Obsidian or a similar readable workspace is treated as a projection and correction surface, not the canonical runtime. The paper defines the layers, core records, state transitions, interfaces, invariants, evaluation program, and failure modes needed before broad autonomy is introduced. The architecture is intended for implementers who need governed business agents that can explain their evidence, stop when authority is missing, produce run receipts, and propose improvement without self-approving consequential change.

The problem

A company does not run on documents alone. It runs on current facts, disputed claims, source authority, goals, constraints, approvals, decisions, workflows, and outcome records. A retrieval system can surface a paragraph, but it cannot reliably decide whether the paragraph is current, whether a later correction superseded it, whether the user has authority to act on it, or whether the proposed action is reversible. A workflow engine can execute steps, but it cannot prove that the job produced the expected business artifact unless completion is defined separately from process exit. An optimizer can recommend a change, but it can also optimize a local proxy while harming a guardrail unless experiments and approvals are explicitly modeled.

The operational failure is not just "bad prompts." It is collapsed state. Source state, belief state, approval state, execution state, and learning state are often blended into one generated answer. That makes the system hard to audit and easy to over-trust. A generated inference may be treated like a company fact. An approval may be treated like execution. A successful API response may be treated like business completion. A local metric gain may be treated like evidence of overall value. The Company Cortex design addresses these categories by forcing separate records, states, and transitions.

Contributions

This paper contributes four implementation-level artifacts.

First, it defines a five-layer architecture for governed business AI: Company Memory, Executive Control, Telemetry, Execution, and Improvement Controller.

Second, it defines cross-cutting authority and tenant controls that bind approvals to exact actions, versions, scopes, approvers, and expiry. Unknown consequential authority fails closed and escalates.

Third, it specifies the role of an Obsidian-style human projection. The projection is useful for review and correction, but direct edits create proposals rather than silent canonical mutations.

Fourth, it gives concrete interfaces, states, invariants, and acceptance criteria that distinguish an implemented architecture from a persuasive diagram.

Core definitions

Company Cortex means the combined runtime and human interface that reconstructs company state, selects governed decisions, executes approved work, records outcomes, and proposes bounded improvements.

Company Memory is the canonical record of sources, assertions, entities, relationships, decisions, versions, conflicts, restrictions, unknowns, and learning records. It is not a folder or a vector index.

Executive Control is the layer that evaluates objectives, constraints, priorities, missing evidence, authority, reversibility, and next best actions or questions.

Telemetry is the event, trace, metric, cost, approval, blocker, and outcome record layer used to understand what happened.

Execution is the versioned workflow and agent layer. It includes simulation, idempotency, pause, retry, receipt, verification, and rollback behavior.

Improvement Controller is the bounded learning loop that proposes hypotheses, experiments, learning records, and change proposals. It cannot approve its own policy, permission, claim, or consequential workflow changes.

Canonical runtime means structured records, append-only events, retained artifacts, and disposable search projections. The runtime, not an editable note, determines current state.

Reference architecture

The Company Cortex is organized around five layers with strict boundaries.

Layer	Primary objects	Primary decisions	Required outputs
Company Memory	sources, assertions, entities, relations, conflicts, versions, restrictions	What is known, unknown, stale, disputed, or superseded?	state snapshot with provenance and exclusions
Executive Control	objectives, constraints, options, questions, priorities, decision packets	What should be done, blocked, escalated, or asked next?	recommendation or inhibition with authority path
Telemetry	events, traces, metrics, costs, blockers, approvals, receipts	What happened, when, why, under which version?	reconstructable run and decision history
Execution	workflows, runs, adapters, simulations, artifacts, rollback records	Can this approved version run, and did it complete?	verified artifact and action receipt contract
Improvement Controller	baselines, hypotheses, experiments, guardrails, learning, change proposals	Should a bounded change be tested or proposed?	evaluated experiment and approved or rejected change

The data flow begins with approved source ingestion. A source record includes owner, classification, effective time, review status, restrictions, and retained evidence. Memory assertions are derived from sources but keep epistemic status separate from source text. The assertion state machine is:

```
proposed -> active -> stale -> superseded
             |-> disputed -> active | superseded | retracted
             |-> retracted
```

The Executive Control layer requests a state snapshot for an objective and effective time. A snapshot contains active assertions, conflicts, unknowns, stale exclusions, restriction notices, and an integrity digest. The controller then builds a decision packet. The packet must state the objective, options, evidence, counterevidence, constraints, authority classification, reversibility, expected artifacts, and recommended next move.

The Execution layer never receives vague permission such as "go ahead." It receives an action-bound authorization decision or an explicit simulation request. Mutating calls require idempotency keys and optimistic version checks. A run can move through:

```
requested -> started -> output_created -> verification -> verified_complete
                |-> blocked -> resumed
                |-> failed -> retry | cancelled | rollback
```

A process exit code is insufficient. A run is complete only when the expected artifact and action receipt contract passes.

The Telemetry layer records events across all other layers. OpenTelemetry's specification provides useful vocabulary for traces, metrics, and logs, but SalesFusion-specific business evidence must be represented in domain records, not just infrastructure spans. OpenTelemetry is therefore contextual support for observability design, not evidence that this architecture produces any business result.

The Improvement Controller closes the loop. It reads telemetry and outcome records, proposes hypotheses, defines experiments, enforces guardrails, evaluates results, and creates change proposals. It does not directly

activate new policies, permissions, claims, or consequential workflow versions. ISO/IEC 42001 and the NIST AI Risk Management Framework support the need for management-system thinking and risk framing, but they do not validate this specific implementation.

Cross-cutting authority and tenant controls

Authority and tenant boundaries are not a sixth layer. They apply to every layer. Every object carries tenant scope. Every request is authenticated. Resource-scoped access follows the least-trust reasoning associated with zero trust architecture: the system evaluates each access attempt against identity, resource, policy, and context rather than trusting broad network location.

Consequence classes define the required path:

Class	Examples	Default path
Read-only	retrieval, summary, simulation	allow if scoped and logged
Internal reversible	draft, internal routing, proposed record	allow with version checks
External reversible	limited action that can be undone	require policy decision and receipt
External consequential	external contact, publication, production mutation	require explicit approval
Financial, legal, security-sensitive	spend, contract, permission, identity	stricter review and expiry

Approval is a record, not an action. It includes action digest, workflow version, scope, approver role, expiry, and conditions. Execution is a separate state transition performed by a policy enforcement point. This separation is designed to block approval replay, approval/execution collapse, and accidental mutation after a changed workflow version; those controls still require adversarial tests.

Obsidian projection

The architecture supports a human-readable projection such as an Obsidian vault because people need to inspect, correct, and reason about company state. The projection contains source summaries, assertion pages, entity pages, conflict sets, decisions, run reports, learning records, and review queues. It is intentionally readable and portable.

However, the projection is not the sole canonical runtime. Direct edits become import proposals. A proposal must pass schema validation, source checks, authority checks, and version checks before canonical state changes. This design protects against accidental bypass while preserving a useful authoring surface. Clients should be able to export their sources, state, policies, decisions, and artifacts without depending on a proprietary note layout.

Implementation

A reference implementation can be decomposed into cortex-api, workers, stores, and interfaces without committing to one vendor stack. The stable contract matters more than the framework.

Core API families include sources, assertions, state snapshots, objectives, policies, approvals, decisions, workflows, runs, experiments, audit, export, and projections. HTTP semantics from RFC 9110 are useful for method behavior, status handling, and cache awareness, while domain correctness still depends on explicit event and state records.

Required data objects include:

Object	Required fields
Source	source id, tenant, owner, classification, effective time, review state, restrictions, version
Assertion	subject, predicate, value, epistemic state, source refs, confidence, effective interval, restriction refs
Decision packet	objective refs, state snapshot, options, constraints, recommendation, authority result, reversibility
Run record	workflow version, inputs, idempotency key, events, artifacts, receipts, blocker, verification result
Experiment	baseline, hypothesis, intervention version, success, failure, guardrails, stop, rollback

The main control loop is deliberately conservative:

```
snapshot = build_state_snapshot(objective, effective_time, freshness, restrictions)
if snapshot.has_unresolved_required_conflict:
    return decision("block", reason="CONFLICT_UNRESOLVED")
if snapshot.missing_required_evidence:
    return decision("collect_evidence", questions=snapshot.unknowns)

packet = compose_decision_packet(snapshot, objective_tree, candidate_actions)
authority = evaluate_policy(packet.proposed_action, actor, tenant, version)
if authority.decision == "deny":
    return decision("block", reason="POLICY_DENIED")
if authority.decision == "require_approval":
    return decision("escalate", approval_request=authority.requirements)

if packet.proposed_action.is_mutating:
    require_idempotency_key()
    require_action_receipt_contract()
return decision("ready_for_execution", packet=packet)
```

The implementation invariant is simple: generated analysis can recommend, propose, classify, or ask. It cannot silently become company fact, policy, approval, or proof of completion.

Evaluation

Evaluation should begin before field claims. The architecture should pass contract, scenario, and adversarial tests before any outcome study is attempted.

Contract correctness tests check schema validity, required tenant and provenance fields, legal state transitions, idempotency behavior, version conflicts, approval expiry, and export compatibility.

Memory quality tests include authoritative facts, contradictory sources, stale sources, generated assertions misrepresented as facts, retracted sources, corrected sources, restricted material requested for public output, and prompt injection inside source text.

Executive-control tests include objective conflict, missing evidence, irreversible requests, expired approvals, low-value activity, urgent requests outside authority, and cases where the correct next move is a question or no action.

Workflow reliability tests measure verified completion, false completion, duplicate work, blocker classification, artifact validity, retry recovery, pause/resume behavior, and rollback proof. The target for unauthorized action and cross-tenant leakage in pre-installation tests is zero.

Outcome evaluation comes later. A business-outcome paper would need baseline, sample, comparison design, measurement window, confounders, null and negative results, and disclosed limitations. This architecture report does not provide those results.

Safety, risks, and failure modes

The major risks are source prompt injection, memory poisoning, silent staleness, conflict erasure, retrieval laundering, false authority, approval replay, duplicate consequential action, false completion, proxy gaming, optimizer privilege escalation, cross-tenant leakage, and export leakage. OWASP's LLM application threat material is relevant to threat awareness, especially prompt injection and unsafe tool use, but the actual control set must be tested in the deployed system.

Several design rules reduce blast radius. Sources are treated as data, not instructions. Search projections are disposable and cannot override canonical state. Conflicting assertions coexist until explicit resolution. Unknown authority fails closed. Approval does not execute. Idempotency keys and action receipts protect retries. The optimizer may propose changes but cannot authorize itself. Obsidian edits become proposals.

Residual risk remains. A human can approve the wrong thing. A source can be deceptive. A connector can drift. A model can produce plausible but wrong analysis. A metric can be corrupted by bad joins or delayed effects. The architecture's purpose is not to remove uncertainty; it is to make uncertainty, authority, execution, and change inspectable.

Limitations

This paper defines a reference architecture, not a completed empirical evaluation. It presents no measured effect on revenue operations, risk, time, or decision quality. It does not specify a full production schema, a complete connector library, a user-interface design, or a formal verification model. The five-layer separation may introduce implementation overhead for small teams. Some organizations may not have source quality, authority clarity, or operational discipline sufficient to benefit from the full design without prior cleanup.

The architecture also depends on correct policy modeling. A machine-readable authority registry is only as accurate as the humans who maintain it. Exportability and tenant isolation require disciplined engineering

across storage, search, artifacts, logs, and support tooling.

Practical implications

An implementer should start with the kernel rather than broad autonomy: approved sources, assertions, conflict sets, state snapshots, authority evaluation, one non-mutating workflow, and evidence-producing run records. The first installation should show that a human can trace every current-state field to a source, see stale and conflicting records, inspect an approval decision, run one proof workflow, and export the relevant state.

The design also changes product planning. A chatbot interface is not the system. A vector index is not memory. A workflow run is not verified completion. A dashboard metric is not a learning record. Each must be represented in the layer where it belongs.

References used

This architecture uses vocabulary and framing from W3C PROV-O for provenance relationships, NIST AI Risk Management Framework for risk-management framing, NIST SP 800-207 for resource-scoped access reasoning, OpenTelemetry Specification for observability concepts, OWASP Top 10 for Large Language Model Applications for threat awareness, ISO/IEC 42001:2023 for AI management-system context, and RFC 9110 for HTTP interface semantics. These references do not validate SalesFusion-specific effectiveness claims.

Conclusion

The Company Cortex is a systems architecture proposal for governed company cognition. Its core move is separation: memory from retrieval, recommendation from approval, approval from execution, process exit from completion, and optimization proposal from authorized change. SalesFusion has implemented selected patterns that motivate the design, but the complete reference architecture still requires implementation, scenario testing, safety testing, and field evaluation before stronger claims are appropriate. The practical starting point is narrow: build source-backed Company Memory, evaluate explicit authority, run one non-mutating job with evidence, and make every consequential next step inspectable.

Changelog

- 2026-07-17 — Version 1.0.0: founder-approved first public release; publication metadata, evidence-status box, limitations, and verified references retained.

SALESFUSION

Governed Continual Improvement Loops

Technical architecture and implementation report

PUBLIC RELEASE · VERSION 1.0.0 · 2026-07-17

Governed Continual Improvement Loops

Publication metadata

Field	Value
Author	SalesFusion Research
Publication date	2026-07-17
Version	1.0.0
Canonical slug	governed-continual-improvement-loops
Canonical path	/technical-papers/governed-continual-improvement-loops
Publication class	Technical architecture and implementation report
Approval state	Founder-approved public release

Revenue Cortex connection

Revenue Cortex treats growth as a governed improvement loop, not a one-time automation build. Signals, campaigns, objections, conversion evidence, sales-call notes, pipeline outcomes, and delivery proof can all create hypotheses, but changes to positioning, workflows, offers, and outreach require versioned experiments, approval, rollback, and evidence review.

Evidence-status box

> This paper proposes a governed continual-improvement protocol for business AI systems. SalesFusion has implemented cadence components and design artifacts for review, reporting, learning, and change proposals, but the complete optimization controller described here has not been evaluated in a longitudinal field study. The allowed claim is protocol-level: business AI optimization should operate through versioned hypotheses, bounded experiments, approvals, outcome measurement, rollback, and regression monitoring. No claim is made that the protocol improves business outcomes, model quality, revenue, or efficiency.

Abstract

Business AI systems should learn from operation, but uncontrolled self-modification creates authority, safety, and measurement problems. A system that can change its own policies, permissions, claims, or consequential workflow behavior can optimize the wrong proxy, hide regression, or expand scope without human authority.

We propose a governed continual-improvement loop that separates telemetry, hypothesis generation, experiment design, approval, execution, evaluation, learning, and change activation. The loop requires baselines, success criteria, failure criteria, guardrails, stop conditions, rollback plans, and regression tests before an experiment starts. Completed experiments create learning records and change proposals; they do not directly promote themselves into production policy or workflow changes. The protocol is intended for business agents whose actions affect operating workflows and therefore require conservative evidence handling, explicit approval, and inspectable failure states.

The problem

Most automation systems degrade if they never adapt. Sources change, connectors drift, objectives shift, and workflow designs become stale. At the same time, a self-improving business agent can create a different failure class. It may optimize a visible local metric while harming a more important downstream metric. It may treat correlation as causation. It may run experiments without a baseline. It may continue after a guardrail breach because the local score still looks good. It may recommend removing the very control that would stop it. It may approve its own change because the optimization loop is wired too close to the execution layer.

The core issue is not whether learning should exist. It is whether learning is governed. A safe operating loop must preserve the difference between observation, hypothesis, experiment, result, learning record, change proposal, approval, deployment, rollback, and regression monitoring. If those states collapse into "the system learned and updated itself," the organization loses the ability to audit what changed and why.

Contributions

This paper contributes a protocol for governed continual improvement.

First, it defines experiment records with required baseline, hypothesis, intervention, success, failure, guardrail, stop, and rollback fields.

Second, it separates experiment approval from change approval. An experiment may be allowed without granting permission to activate a consequential workflow change.

Third, it specifies controls for proxy gaming, causal overreach, self-authorization, and regression.

Fourth, it defines implementation interfaces and state transitions for experiments, learning records, and change proposals.

Fifth, it describes an evaluation program for optimization integrity before business-outcome claims are made.

Definitions

Baseline is the pre-intervention state of a metric, workflow, population, or process during a defined time window.

Hypothesis is a testable statement about how an intervention is expected to affect defined metrics under stated conditions.

Experiment is a bounded change to a workflow, policy-adjacent behavior, ranking rule, prompt, interface, or process, run under predeclared criteria and authority.

Guardrail is a metric, rule, or condition that must not be harmed or breached, even if the primary metric moves favorably.

Stop condition is a predeclared rule that pauses or terminates an experiment.

Rollback is the controlled return to a prior approved version or state, with proof that the reversion occurred.

Learning record is a post-evaluation artifact that states what was observed, evidence strength, limits, confounders, and applicability.

Change proposal is a request to modify memory, workflow, policy, permission, or operating cadence. It requires approval appropriate to its consequence class.

Governed improvement protocol

The protocol has seven stages:

Stage	Primary object	Exit condition
observe	telemetry and outcome records	anomaly, opportunity, or failure pattern documented
hypothesize	hypothesis record	testable statement and scope written
design	experiment record	baseline, metrics, guardrails, stop, rollback complete
approve	approval record	authority approves exact experiment scope
run	experiment events	observations collected under versioned intervention
evaluate	learning record	result classified with limits and confounders
propose change	change proposal	separate approval path for durable change

The experiment record should include:

```
{
  "experiment_id": "exp_guardrail_review_001",
  "objective_refs": ["obj_operating_reliability"],
  "baseline_window": "defined_period",
  "hypothesis": "If intervention X is applied to scope Y, metric A will move without guardrail B
breach.",
  "intervention_version": "workflow_v4_candidate",
  "population_scope": "bounded_group",
  "success_criteria": ["primary_metric_threshold"],
  "failure_criteria": ["no_valid_measurement", "negative_primary_result"],
  "guardrails": ["quality_floor", "authority_error_zero", "complaint_threshold"],
  "stop_conditions": ["guardrail_breach", "missing_data", "connector_error"],
  "rollback_plan": "return_to_workflow_v3_and_verify_receipts",
  "approval_required": ["business_owner", "operator"],
  "self_authorization_allowed": false
}
```

The Improvement Controller may propose this record. It cannot approve it. It also cannot approve a later change proposal that modifies policy, permission, claim state, or consequential workflow behavior. This is a hard invariant.

Baselines, hypotheses, and experiment design

A baseline is not a casual memory of how things usually work. It must define time window, population, source, metric semantics, missing-data treatment, and known concurrent changes. If the baseline is weak, the experiment should be labeled exploratory and should not produce a strong conclusion.

Hypotheses should be narrow. A useful hypothesis states the expected direction of a primary metric, the affected scope, the time window, and the guardrails. "Make the workflow better" is not an experiment. "Change step order for this non-mutating analysis job and compare verified artifact completion under the same input fixture" is closer to a testable design.

Experiment design must record comparison logic. Depending on risk and feasibility, a team might use before/after comparison, holdout, canary, shadow mode, simulation, or expert scoring. The method should match consequence class. A read-only ranking change may be evaluated in simulation. A workflow that mutates an external system may require canary, explicit approval, and rollback proof before broader use.

Guardrails, stop conditions, rollback, and regression

Guardrails are intended to detect local optimization that violates system-level constraints. Examples include unauthorized-action rate, cross-tenant leakage rate, false-completion rate, complaint threshold, error budget, blocker classification quality, output restriction compliance, and downstream outcome preservation. A local metric gain with guardrail breach is failure by definition.

Stop conditions must be executable. They should specify the event, threshold, actor, and required response. A stop condition such as "if things look risky" is too vague. A better condition is: if an experiment produces any unauthorized external action, stop immediately, create a safety event, roll back the workflow version, and escalate to the authority owner.

Rollback must be verified, not assumed. A rollback record should include prior version, target version, affected scope, execution receipts, state reconciliation, unresolved side effects, and human review status. If rollback is unavailable or unsafe, that fact must be known before the experiment starts.

Regression monitoring continues after a change is approved. The system should run scenario benchmarks against prior objectives, guardrails, prompt-injection tests, authority tests, and model/version changes. OpenTelemetry concepts are useful for traces, metrics, and logs, but domain regression requires business-state records as well as infrastructure telemetry.

Preventing proxy gaming and self-authorization

Proxy gaming occurs when an experiment improves what is measured while harming what matters. The protocol addresses this with objective trees, anti-goals, guardrails, delayed-outcome review, and human evaluation of applicability limits. It also records negative, null, blocked, and failed experiments so the system does not learn only from attractive summaries.

Self-authorization is a separate risk. The Improvement Controller is allowed to:

Allowed	Not allowed
identify anomalies	approve its own experiment
propose hypotheses	change authority policy
draft experiment records	expand connector permissions
evaluate observations	activate consequential workflow version
create learning records	promote correlation to company fact
draft change proposals	delete guardrails or restrictions

This separation reflects the same governance posture used elsewhere in the Company Cortex: generated analysis may propose, but authority changes require human approval and auditable events.

Implementation

The protocol can be implemented with event-backed records and stable endpoints:

```
POST /v1/experiments
POST /v1/experiments/{experiment_id}/approve
POST /v1/experiments/{experiment_id}/start
POST /v1/experiments/{experiment_id}/observe
POST /v1/experiments/{experiment_id}/stop
POST /v1/experiments/{experiment_id}/evaluate
POST /v1/experiments/{experiment_id}/rollback
POST /v1/learning-records
POST /v1/change-proposals
```

The experiment state machine is:

```
draft -> evidence_needed -> approval_needed -> approved -> running
                                                 |-> paused -> running
                                                 |-> completed -> evaluated
                                                 |-> failed
                                                 |-> stopped_for_safety -> rollback
```

Events include experiment.proposed, experiment.approved, experiment.started, experiment.observation_recorded, experiment.guardrail_breached, experiment.stopped, experiment.completed, experiment.evaluated, experiment.rollback_started, experiment.rolled_back, learning_record.created, change_proposal.created, change_proposal.approved, change_proposal.rejected, and regression.detected.

SalesFusion has implemented cadence components in delivery design: weekly review of run health and blockers, monthly review of constraints and experiments, quarterly review of objectives and permissions, and learning fields in reporting. The complete controller still needs a reference implementation and longitudinal evaluation.

Evaluation

Optimization integrity should be evaluated before outcome effectiveness. Each experiment can be scored for pre-registered hypothesis, baseline validity, comparison design, intervention version, success and failure criteria, guardrail coverage, stop conditions, rollback feasibility, data completeness, confounder recording, conclusion strength, applicability limits, and approved change event.

Adversarial cases should include apparent local improvement with worse downstream outcomes, missing data, seasonality, changing population, delayed outcomes, model changes during the test, optimizer proposal to remove a guardrail, and a workflow that tries to approve its own permission expansion.

Acceptance criteria:

Criterion	Required behavior
baseline completeness	experiment cannot start without required baseline fields
guardrail breach	experiment stops and creates rollback/escalation events
self-authorization	optimizer cannot approve own policy or permission change
causal discipline	result language matches comparison strength
rollback proof	prior version restoration is verified
regression detection	approved change is monitored after activation

A later field evaluation would need multiple installations, visible baselines, time windows, confounders, negative results, and limitations. This paper does not provide that evidence.

Safety, risks, and failure modes

The main risks are proxy gaming, causal overreach, data corruption, hidden confounders, unsafe irreversible experiments, optimizer privilege escalation, rollback failure, regression after deployment, and selective reporting. NIST AI RMF and ISO/IEC 42001 support governance and management-system framing, but they do not validate this protocol's effectiveness.

The protocol fails closed where possible. Missing baseline blocks launch. Missing rollback blocks or escalates high-consequence experiments. Guardrail breach stops the run. Authority ambiguity escalates. Change activation requires a separate approval event. Result uncertainty is recorded rather than polished away.

Limitations

The protocol increases process cost. Small changes may feel slow when every experiment needs baseline, guardrails, and rollback. Some workflows lack clean comparison groups. Some outcomes are delayed, sparse, or affected by external events. Human reviewers can still approve bad experiments or misread results. The controller also depends on the quality of telemetry and outcome ledgers; corrupted metrics produce corrupted learning.

This paper does not claim that governed improvement loops outperform manual review or unguided optimization. It defines an inspection structure intended to make optimization risks reviewable and specifies the tests needed to evaluate it.

Practical implications

Teams should start with proposal-only optimization. Let the system identify blockers, draft hypotheses, and prepare experiment records. Require human approval for experiment launch and separate approval for durable change. Run early experiments in simulation or non-mutating mode where possible. Preserve failed and null results because they are part of the learning system.

The operating question should shift from "did the agent improve itself?" to "what changed, under which hypothesis, against which baseline, with which guardrails, and who approved the durable change?"

References used

This paper uses NIST AI Risk Management Framework for risk framing, ISO/IEC 42001:2023 for AI management-system context, OpenTelemetry Specification for observability concepts, and OWASP Top 10 for Large Language Model Applications for threat awareness. These references do not support SalesFusion-specific effectiveness claims.

Conclusion

Continual improvement is useful only if it remains governed. The Company Cortex improvement loop separates observation from hypothesis, experiment from activation, and learning from authority. SalesFusion has

implemented cadence components, while the full optimization controller remains a proposal requiring implementation and longitudinal evaluation. The central rule is non-negotiable: the optimizer may recommend and experiment under approval, but it cannot self-authorize consequential change.

Changelog

- 2026-07-17 — Version 1.0.0: founder-approved first public release; publication metadata, evidence-status box, limitations, and verified references retained.

SALESFUSION

Executive Control for Business Agents

Technical architecture and implementation report

PUBLIC RELEASE · VERSION 1.0.0 · 2026-07-17

Executive Control for Business Agents

Publication metadata

Field	Value
Author	SalesFusion Research
Publication date	2026-07-17
Version	1.0.0
Canonical slug	executive-control-for-business-agents
Canonical path	/technical-papers/executive-control-for-business-agents
Publication class	Technical architecture and implementation report
Approval state	Founder-approved public release

Revenue Cortex connection

Revenue Cortex uses executive control to decide what revenue work should run, pause, escalate, or remain blocked. In a growth system, this includes selecting the next buyer segment, approving outbound claims, routing sales opportunities, holding unsafe automation, and preserving human authority over consequential customer-facing actions.

Evidence-status box

> This paper proposes an executive-control layer for business agents and describes selected SalesFusion controls that have been implemented as operating patterns: authority boundaries, approval separation, decision records, and first-job gates. The complete executive-control architecture requires expert-scored scenario evaluation before claims about decision quality or operational effectiveness are appropriate. The allowed claim is architectural: business agents need an executive-control layer that prioritizes, inhibits, escalates, and allocates work under explicit authority and constraints.

Abstract

Business agents are often evaluated by whether they can complete a task. In operating environments, task completion is not enough. The harder question is whether the agent should attempt the task, under which objective, with which evidence, within which authority, after asking which question, and with what rollback path. We define executive control as a layer between Company Memory and workflow execution. It builds objective

trees, detects constraints, ranks options, selects next best actions or questions, inhibits unsafe or low-evidence work, escalates authority gaps, and produces decision packets. The layer treats do_nothing, collect_evidence, block, and escalate as valid outputs, not failures. This paper specifies data objects, priority rules, inhibition logic, authority checks, interfaces, evaluation scenarios, and failure modes for executive control in governed business AI systems.

The problem

An agent with tools can act faster than a team can review every intermediate step. That speed is useful only when the agent can decide when not to act. Business settings contain conflicting goals, stale context, missing evidence, irreversible actions, external consequences, approvals that expire, and tasks that create activity without advancing an objective. A workflow runner cannot resolve those issues alone. A language model prompt can list rules, but prompts are not durable governance.

The common failure is direct task execution. A request enters the system and the agent tries to satisfy it. The agent may ignore the current objective, choose a visible but low-value task, ask the wrong human for approval, reuse stale context, or continue a run after the correct answer is to stop. When recommendations and execution are collapsed, the system has no clean place to inspect tradeoffs or enforce authority.

Executive control is the missing layer. It is not a manager persona. It is a stateful control system that converts memory, objectives, constraints, and policies into a decision packet before execution.

Contributions

This paper contributes a technical model for executive control in business agents.

First, it defines objective trees, constraints, anti-goals, priority factors, authority classifications, and decision packets.

Second, it specifies next-best-action and next-best-question selection as separate outcomes.

Third, it defines inhibition and escalation rules that block action under missing evidence, unresolved conflict, stale context, duplicate work, insufficient authority, or unsafe reversibility.

Fourth, it describes implementation interfaces connecting memory snapshots, policy evaluation, decisions, approvals, workflows, and telemetry.

Fifth, it proposes an evaluation suite for constrained business decisions without claiming field results.

Definitions

Objective tree is a hierarchy of desired outcomes, subgoals, metrics, constraints, anti-goals, and tradeoff rules.

Constraint is a binding condition that limits action. Constraints can be policy, capacity, legal, financial, security, data-quality, timing, or objective-conflict constraints.

Priority is the controller's ordering of candidate work under objective value, urgency, evidence quality, reversibility, dependency, cost, risk, and authority.

Next best action is the highest-ranked allowed action after constraints, evidence, and authority checks.

Next best question is the highest-value question when action is blocked by missing, conflicting, stale, or restricted information.

Inhibition is an explicit decision to prevent or delay an action.

Escalation is routing a decision to a human or role with the required authority and context.

Decision packet is the inspectable artifact that contains objective, state snapshot, evidence, options, constraints, recommendation, authority result, expected artifacts, reversibility, and confidence.

Executive-control model

The controller receives a request or trigger and does not immediately execute it. It performs six operations:

1. Build a state snapshot for the relevant objective, effective time, freshness requirement, and actor.
2. Resolve the objective tree and identify constraints, anti-goals, dependencies, and metric definitions.
3. Generate options, including non-action options such as do_nothing, collect_evidence, block, and escalate.
4. Score options using transparent factors and reject options that violate hard constraints.
5. Evaluate authority and approval requirements for the selected option.
6. Produce a decision packet or route an approval request.

A minimal objective tree:

```
{
  "objective_id": "obj_operating_motion_quality",
  "outcome": "improve an approved operating motion",
  "metrics": ["metric_quality", "metric_cycle_time"],
  "constraints": ["no_external_mutation_without_approval"],
  "anti_goals": ["activity_without_outcome_signal", "policy_bypass"],
  "priority_rules": [
    "hard_constraints_override_score",
    "fresh_source_required_for_consequential_action",
    "irreversible_action_requires_escalation"
  ]
}
```

A candidate option can be represented as:

```
{
  "option_id": "opt_collect_evidence",
  "type": "collect_evidence",
  "objective_refs": ["obj_operating_motion_quality"],
  "required_assertions": ["asrt_current_constraint"],
  "expected_artifact": "updated_state_snapshot",
  "consequence_class": "read_only",
  "reversibility": "not_applicable",
  "authority_required": "analyst",
  "blocked_by": []
}
```

Priority is not a single opaque score. The controller should preserve factor values:

Factor	Question
objective value	Which objective or constraint does this advance?
urgency	Does delay change the state or opportunity?
evidence quality	Are required assertions active, fresh, and sourced?
reversibility	Can the action be undone and proven undone?
authority	Is policy clear and approval current?
dependency	Does another task need this result?
cost and capacity	Is the work bounded by budget and operator capacity?
risk	What is the downside of wrong execution?

Hard constraints override score. If a highly valuable action lacks authority, the answer is escalation, not execution.

Inhibition, escalation, and decision rules

Inhibition rules are explicit and testable:

Condition	Controller output
required source is stale	collect evidence
required assertion has unresolved conflict	block or escalate
action is outside actor authority	escalate or deny
approval expired or version-mismatched	require new approval
action is duplicate of a completed receipt	return prior receipt
action is irreversible and rollback absent	escalate before execution
objective conflict is unresolved	decision needed
no action is warranted	do_nothing with reason

The control loop:

```
snapshot = memory.snapshot(objective, actor, effective_time)
constraints = resolve_constraints(objective_tree, snapshot, policies)
options = generate_options(request, objective_tree, snapshot)

for option in options:
    option.blockers = detect_blockers(option, constraints, snapshot)
    option.authority = policy.evaluate(option.action_digest, actor, scope)
    option.score = score_priority(option, objective_tree)

eligible = options.without_hard_blockers()
if eligible.is_empty():
    return packet(output=best_inhibition(options))

selected = rank(eligible).first()
if selected.authority.requires_approval:
    return packet(output="escalate", approval_request=selected.authority)
return packet(output="ready", selected=selected)
```

The controller must explain why a rejected option was rejected. Without rejection reasons, a reviewer cannot distinguish good inhibition from model hesitation.

Authority and decision packets

Authority is evaluated through policy, not inferred from user tone. The system classifies actions as read-only, internal reversible, external reversible, external consequential, financial, legal, or security-sensitive. Unknown classification defaults to the stricter path and escalates.

Approval is bound to exact action, version, scope, approver, and expiry. It does not execute the action. Recommendation and decision are separate operations and may require separate principals. This pattern is consistent with risk framing from NIST AI RMF and resource-scoped access reasoning from NIST SP 800-207, but the references do not validate any SalesFusion-specific outcome.

A decision packet should include:

Field	Required content
objective context	objective refs, metrics, anti-goals
memory state	snapshot id, evidence, conflicts, unknowns, exclusions
options	considered actions and questions
recommendation	selected next move and rationale
inhibition log	rejected options and blockers
authority	policy result, approval requirement, consequence class
execution contract	expected artifact, receipt, idempotency key requirement
reversibility	rollback path or escalation reason
confidence	calibrated statement and limits

Decision packets are the main artifact of executive control. They let a human inspect the reasoning without reading every retrieved source or the full model transcript.

Implementation

The executive-control layer can be implemented as a service that sits between state snapshots and workflow runs. It should use stable APIs:

```
POST /v1/state/snapshot
POST /v1/policies/evaluate
POST /v1/decisions
POST /v1/decisions/{decision_id}/evidence
POST /v1/decisions/{decision_id}/recommend
POST /v1/approvals
POST /v1/workflows/{workflow_id}/simulate
POST /v1/workflows/{workflow_id}/runs
```

SalesFusion has implemented selected controls in its operating patterns: approval boundaries, first-job gates, run evidence, and decision-oriented delivery records. The full controller should add objective-tree resolution, transparent priority scoring, systematic inhibition, and expert-reviewable packets.

Implementation invariants:

Invariant	Reason
no source-free consequential recommendation	blocks generated fact from driving action by rule
no execution from approval event alone	blocks approval/execution collapse by rule
no policy bypass from prompt or source text	keeps instructions out of evidence
no hidden overwrite of objective or constraints	preserves reviewability
no completion without receipt contract	checks for the evidence required to reject false-completion claims

The controller should also log all decision events with trace IDs. OpenTelemetry concepts can support infrastructure observability, while domain events record decision evidence, approvals, blockers, and outcomes.

Evaluation

Executive-control evaluation should use scenario tasks scored by domain experts. Scenarios should include several valid tasks but one binding constraint, objective conflict, missing evidence, irreversible request, expired approval, low-value work with high activity, urgent request outside authority, and a case where no action is warranted.

Scoring dimensions:

Dimension	What reviewers inspect
objective alignment	recommendation tied to the right objective
constraint identification	hard constraints recognized and enforced
priority quality	option ranking is defensible
option completeness	question, block, and do-nothing options considered
inhibition correctness	unsafe or unsupported actions stopped
escalation correctness	right authority path selected
reversibility awareness	rollback or irreversibility handled
explanation traceability	packet links to sources and policy
calibration	confidence reflects evidence quality

Adversarial tests should include prompt requests to ignore policy, source documents containing instructions, approval references from another version, a principal trying to expand its own scope, and one tenant referencing another tenant's object. Pre-installation target for unauthorized action and cross-tenant leakage should be zero.

This evaluation can support claims about controller behavior on the test suite. It cannot support claims about revenue impact or general decision superiority without a separate field study.

Safety, risks, and failure modes

Executive control can fail by over-acting, under-acting, escalating to the wrong role, ranking proxy activity above outcome-relevant work, treating an approval as execution, accepting stale evidence, suppressing uncertainty, or producing packets that are too vague to review. It can also become a single point of policy error if the authority registry is wrong.

Controls include explicit consequence classes, fail-closed unknown authority, approval expiry, version binding, inhibition logs, separate recommendation and execution events, duplicate-action detection, and review queues. OWASP LLM application threats are relevant to prompt injection and tool misuse; the specific controls must be tested in context.

Human error remains. The controller can make a recommendation inspectable, but a human can still approve a poor action. The system should therefore preserve rejected, blocked, and deferred decisions for later review rather than only celebrating completed work.

Limitations

This paper does not present a validated benchmark result. It defines the controller and the evaluation needed. Priority scoring remains partly judgment-based until enough labeled decisions exist. Objective trees can be incomplete or internally inconsistent. Authority policy can lag real organizational responsibility. Some actions have ambiguous reversibility, and some outcomes are delayed enough that the controller cannot know whether a decision was good within the run window.

The architecture also assumes the memory layer can provide source-backed state. If Company Memory is weak, executive control will mostly produce better-labeled uncertainty rather than better actions.

Practical implications

Builders should place executive control before workflow execution. The first version does not need a complex optimizer. It needs a state snapshot, objective tree, consequence classifier, policy evaluation, option generator, inhibition rules, and decision packet.

Operators should expect the controller to ask questions and block work. A system that always returns an action is not exercising executive control. A good first deployment should include read-only scenarios, simulated runs, approval tests, and explicit no-action cases.

References used

This paper uses NIST AI Risk Management Framework for risk framing, NIST SP 800-207 for resource-scoped access reasoning, OpenTelemetry Specification for observability concepts, and OWASP Top 10 for Large

Language Model Applications for threat awareness. These references do not validate SalesFusion-specific executive-control effectiveness.

Conclusion

Business agents need more than memory and tools. They need a control layer that decides what not to do, what to ask, when to escalate, and how to bind a recommendation to authority and evidence. SalesFusion has implemented selected governance controls that support this direction, while the complete executive-control layer remains a proposed architecture requiring expert-scored evaluation. The practical standard is clear: before a workflow runs, the system should produce a decision packet that a competent reviewer can inspect, challenge, approve, reject, or send back for evidence.

Changelog

- 2026-07-17 — Version 1.0.0: founder-approved first public release; publication metadata, evidence-status box, limitations, and verified references retained.

SALESFUSION

Evidence-Gated Agentic Workflows

Technical architecture and implementation report

PUBLIC RELEASE · VERSION 1.0.0 · 2026-07-17

Evidence-Gated Agentic Workflows

Publication metadata

Field	Value
Author	SalesFusion Research
Publication date	2026-07-17
Version	1.0.0
Canonical slug	evidence-gated-agentic-workflows
Canonical path	/technical-papers/evidence-gated-agentic-workflows
Publication class	Technical architecture and implementation report
Approval state	Founder-approved public release

Revenue Cortex connection

Revenue Cortex uses evidence-gated workflows to keep buyer research, outbound drafts, campaign setup, approval, execution, replies, and outcomes in distinct states. This matters because revenue work becomes risky when an approved idea, a generated draft, a scheduled campaign, and a verified result collapse into the same record.

Evidence-status box

> SalesFusion uses separation of evidence, approval, execution, and outcome records as an operational governance pattern. This paper defines that pattern and its control logic. It does not claim a measured reduction in incidents, policy violations, or review time. Those claims require the false-authorization and policy-adherence evaluation described below.

> This is a public technical architecture and implementation report. It uses provenance and append-only event-history concepts, risk framing consistent with the NIST AI Risk Management Framework, and LLM-application threat awareness consistent with the OWASP Top 10 for Large Language Model Applications. These references support vocabulary and risk framing only; they do not validate SalesFusion-specific effectiveness.

Abstract

Agentic workflows often collapse several different questions into one operational flag: can the system act? That collapse is unsafe for consequential business workflows. A sourced statement may be well proven but not approved for use. A human may approve one action without authorizing a different version, channel, audience, or time window. A workflow may execute after approval but fail to produce the expected artifact. An outcome may be recorded without proving that the action was authorized. Evidence-gated workflows treat these as separate state machines.

We define five state families: provenance state, claim state, approval state, execution state, and outcome state. Paper 12 specializes the authority and approval portion of this broader evidence-gating model. The paper specifies the records, transitions, invariants, and interfaces needed to keep them separate. The central rule is simple: approval is not execution, and execution is not outcome. Approval grants authority to attempt a specific action under a specific scope. Execution records what the system attempted and what receipts or artifacts it produced. Outcome records what happened after the action, including null, blocked, failed, and adverse states. The design is intended for practitioners building AI-assisted business systems where generated analysis can inform work but must not silently become fact, authority, or completion proof.

The problem

The practical failure mode is not usually that a system has no evidence. The failure is that it has the wrong kind of evidence for the decision being made. A source may prove that a company policy exists, but it does not prove that the current request fits that policy. A reviewer may approve a recommendation, but the approval may refer to a prior draft. A workflow may return a successful process exit code, but the external action may not have occurred or may have occurred twice. A metric may move after the run, but the movement may be unrelated, delayed, duplicated, or outside the approved measurement window.

Many workflow systems hide these distinctions behind coarse states such as ready, approved, done, or sent. Those states are convenient for queues but weak for governance. They encourage last-write-wins memory, approval replay, stale context reuse, and false completion. In an AI-assisted workflow, the risk grows because generated summaries can launder uncertainty. A model can present a sourced fact, a plausible inference, and an unsupported completion statement in the same fluent paragraph. If the runtime treats that paragraph as operational state, the boundary between evidence and authority disappears.

Evidence gating solves a narrower but important problem: it gives each consequential transition a typed reason to be allowed, blocked, escalated, retried, or rejected. It does not make the workflow correct by itself. It makes incorrectness easier to locate and harder to convert into an authorized action without leaving a record.

Contributions

This paper contributes a reusable workflow model with five separated state families, an event envelope for transitions, a policy for binding approvals to exact actions, and a verification protocol designed to keep approval, execution, and outcome from substituting for one another.

The implementation contribution is a SalesFusion pattern: source-backed assertions, approval records, workflow runs, action receipts, and outcome reviews are represented as separate records joined by correlation

and causation identifiers. The design contribution is the set of invariants that must hold across those records. The evaluation contribution is a test plan focused on false authorization, stale evidence, approval replay, duplicate action, and false completion.

Core definitions

Provenance state records where an input came from, who or what supplied it, when it was observed, what restrictions apply, and whether it is current, stale, retracted, or superseded. A provenance state can support a claim but cannot approve an action.

Claim state separates epistemic status from lifecycle state. Epistemic status distinguishes observed_fact, authorized_assertion, inference, hypothesis, estimate, and unknown; lifecycle state distinguishes active, stale, disputed, superseded, and retracted. A claim can be eligible as evidence only when its source, freshness, restrictions, and conflict status satisfy the current decision rule.

Approval state records a human or policy decision about a specific action digest, version, scope, actor class, approver role, conditions, and expiry. Approval authorizes an attempt; it does not create the attempt.

Execution state records a workflow run and any external or internal actions attempted by that run. It includes idempotency keys, connector receipts, artifacts, retries, pauses, failures, and rollback records.

Outcome state records the reviewed consequence or result of the run. It may be positive, negative, null, blocked, indeterminate, delayed, or outside the measurement window.

Evidence-gated architecture

The architecture uses separate records connected by references rather than one mutable status field.

State family	Example states	Primary question	Cannot substitute for
Provenance	registered, reviewed, stale, retracted, restricted	Where did this input come from and can it be used?	Claim truth, approval, execution, outcome
Claim	observed_fact, authorized_assertion, inference, hypothesis, estimate, unknown	What is the statement's epistemic position?	Source existence, lifecycle, approval, action receipt
Claim lifecycle	active, stale, disputed, superseded, retracted	Is the claim currently eligible, contested, or replaced?	Epistemic status, approval, execution
Approval	requested, approved, rejected, expired, revoked, invalid	Who authorized exactly what?	Connector call, delivery receipt, result
Execution	requested, started, output_created, failed, rolled_back, verified_complete	What ran and what artifacts exist?	Human authorization, business outcome
Outcome	pending, no_result, accepted, rejected, adverse, inconclusive	What happened after the run?	Source proof, authority, run completion

The workflow event envelope includes tenant_id, event_id, event_type, occurred_at, recorded_at, actor_ref, source_ref, resource_refs, correlation_id, causation_id, trace_id, schema_version, and classification metadata. The envelope is append-only. Current state is reconstructed from events and record versions, not overwritten in place. Provenance relationships follow the vocabulary orientation of PROV-O.

The minimal data objects are:

```
{
  "claim_record": {
    "claim_id": "clm_123",
    "subject_ref": "entity_456",
    "predicate": "has_current_need",
    "object_value": "source-backed value or structured object",
    "epistemic_status": "authorized_assertion",
    "lifecycle_state": "active",
    "evidence_refs": ["src_version_1"],
    "confidence_label": "bounded",
    "effective_at": "2026-07-16T00:00:00Z",
    "restrictions": ["no_public_quote"]
  },
  "approval_record": {
    "approval_id": "app_123",
    "action_digest": "sha256:...",
    "workflow_version": "wf_v7",
    "scope": {"channel": "internal_review", "max_recipients": 1},
    "decision": "approved",
    "conditions": ["use_approved_claims_only"],
    "expires_at": "2026-07-23T00:00:00Z"
  }
}
```

The action digest is computed from the intended action, workflow version, parameters, target resource, consequence class, and required evidence set. If any of those fields change, a prior approval no longer authorizes execution.

Decision rules and invariants

Evidence-gated workflows require explicit invariants. The most important invariant is that a downstream state cannot backfill an upstream state. A successful run cannot make a stale claim current. A human approval cannot make a missing source exist. An outcome review cannot authorize the action that produced the outcome.

Core invariants:

Invariant	Reason
Every claim used for action must reference at least one allowed source version.	Blocks generated assertion from becoming operational fact by rule.
Restricted or stale provenance cannot satisfy a rule requiring current unrestricted evidence.	Checks for source misuse and silent staleness.
Approval must bind to action digest, workflow version, scope, approver role, conditions, and expiry.	Blocks replay against a changed action by rule.
Execution must check approval at start and before each consequential external action.	Checks for stale or revoked authority during a long run.
Verified completion requires expected artifacts and action receipts.	Rejects process exit or model self-report as sufficient completion evidence.
Outcome review must preserve null, blocked, failed, and adverse results.	Makes erasure of negative evidence detectable.

The execution gate is intentionally repetitive:

```
for each consequential step:
  collect required claim_refs
  verify provenance state for each claim_ref
  verify claim state and conflict status
  compute action_digest
  evaluate policy for consequence_class and scope
  require approval if policy says approval_required
  validate approval action_digest, version, conditions, and expiry
  execute with idempotency key
  store receipt or blocker
  verify expected artifact contract
```

Approval cannot equal execution because it answers a different question. Approval says: a qualified principal accepts this exact attempted action under bounded conditions. Execution says: a runtime attempted the action and recorded what happened. The first can exist without the second, for example when a workflow is paused, expired, blocked by missing evidence, or denied by a connector. The second can be attempted only after the first when policy requires it, but it still may fail, duplicate, roll back, or produce no result. Treating approval as execution creates false completion and hides connector failure. Treating execution as approval creates unauthorized action.

Implementation

SalesFusion uses this pattern through a service boundary with source, assertion, policy, approval, decision, workflow, run, audit, and export resources. The specific product stack is not the important part. The important implementation properties are tenant-scoped records, idempotent mutating requests, optimistic version checks, append-only domain events, and deterministic validators for state transitions.

The workflow adapter receives a decision packet rather than a free-form instruction. The packet contains current state, evidence references, unresolved conflicts, required approvals, action class, reversibility, and

expected artifacts. The adapter cannot mark a run complete by returning prose. It must produce the artifact contract: output object, connector receipt where applicable, trace reference, validation result, and blocker classification if the contract is not met.

Approval and execution are different resources. An approval endpoint creates or decides an approval request. A run endpoint starts, pauses, resumes, retries, verifies, or rolls back a workflow run. A policy evaluation endpoint may return deny, allow, or require_approval; it does not perform the external action. The policy enforcement point evaluates immediately before the action and records the result.

For example, a run may have this legitimate path:

```
source.reviewed
assertion.activated
decision.evidence_added
approval.requested
approval.approved
run.requested
run.started
action.authorized
action.started
action.receipt_recorded
run.output_created
run.verified_complete
decision.outcome_reviewed
```

It may also stop safely:

```
approval.approved
run.started
assertion.marked_stale
policy.evaluated = SOURCE_INSUFFICIENT
run.blocked
approval.expired
```

That path is not an operational embarrassment; it is a correct representation of changed evidence.

Evaluation

The primary evaluation is not a business-outcome study. It is a control-correctness and failure-detection suite. A benchmark corpus should include scenarios with stale sources, contradictory assertions, restricted source text, prompt injection inside source content, approval for an old workflow version, approval outside scope, revoked approval, connector failure, retry duplication, missing artifacts, and outcome records with no causal interpretation.

Suggested measures:

Measure	Definition
Unauthorized-action rate	Consequential actions executed without valid policy and approval state.
False-approval rate	System treats invalid, expired, mismatched, or incomplete approval as valid.
False-completion rate	Run marked verified without expected artifacts and receipts.
Duplicate consequential action rate	Retry or event duplication produces more than one external action receipt.
Evidence exclusion accuracy	Stale, restricted, retracted, or disputed claims are excluded when policy requires exclusion.
Traceability completeness	Reviewer can navigate from outcome to run, approval, claim, and source.

The pre-installation target for unauthorized action and cross-tenant leakage should be zero in the adversarial suite. That target is an acceptance criterion, not evidence that future operations will have zero incidents.

Safety and failure modes

The main safety failures are boundary failures. Source prompt injection can attempt to turn source text into instructions. Memory poisoning can promote a false assertion. Retrieval laundering can remove uncertainty from a generated summary. Approval replay can reuse a prior approval for a changed action. Duplicate action can occur when a retry loses the prior receipt. False completion can occur when a process exits successfully but the expected artifact is missing. Outcome inflation can occur when null or adverse results are omitted from review.

Controls include treating sources as data, preserving claim state, requiring current source eligibility, binding approvals to immutable action digests, using idempotency keys, recording independent action receipts, preserving blocker states, and failing closed when authority is unknown. OWASP LLM risk guidance is useful here because source instructions, tool use, sensitive information disclosure, and excessive agency are practical threats in agentic systems.

Limitations

This architecture does not prove that a workflow is commercially useful. It does not make bad sources good, remove human judgment, or ensure that a reviewer understands a decision. It can preserve uncertainty, but it cannot resolve uncertainty without better evidence or authority. It also adds operational cost: more records, more checks, and more explicit blocker states. Teams that only need low-risk internal drafting may find the full pattern excessive.

The current SalesFusion evidence supports saying that the separation pattern is implemented. It does not support policy-violation, revenue, trust, or decision-quality outcome claims. Those remain evaluation questions.

Practical implications

Practitioners should stop asking whether an agentic workflow is simply ready. They should ask which state is ready. Is the source reviewed? Is the claim active and current? Is the approval valid for this exact version and scope? Has execution produced receipts? Has the outcome been reviewed without erasing null or failed results?

This changes product design. Queue labels should expose distinct state families. Approval screens should show action digests in human-readable form. Run pages should show expected artifacts and verification status. Outcome pages should link back to source, claim, approval, and execution records. Generated summaries should be treated as explanations of state, not state itself.

Conclusion

Evidence-gated workflows make consequential agentic work inspectable by separating provenance, claim, approval, execution, and outcome state. The separation is not bureaucratic decoration. It defines controls intended to block a source from becoming authority, an approval from becoming execution, and an execution attempt from becoming proof of outcome. SalesFusion uses this as a governance pattern and proposes a focused evaluation suite for policy adherence and false authorization. The architecture is most useful where AI-assisted workflows touch external parties, business records, publication, permissions, or other consequential surfaces.

References

- W3C, "PROV-O: The PROV Ontology." https://www.w3.org/TR/prov-o/
- National Institute of Standards and Technology, "AI Risk Management Framework." https://www.nist.gov/itl/ai-risk-management-framework
- OWASP Foundation, "OWASP Top 10 for Large Language Model Applications." https://owasp.org/www-project-top-10-for-large-language-model-applications/

Changelog

- 2026-07-17 — Version 1.0.0: founder-approved first public release; publication metadata, evidence-status box, limitations, and verified references retained.

SALESFUSION

Human Authority in Company AI Systems

Technical architecture and implementation report

PUBLIC RELEASE · VERSION 1.0.0 · 2026-07-17

Human Authority in Company AI Systems

Publication metadata

Field	Value
Author	SalesFusion Research
Publication date	2026-07-17
Version	1.0.0
Canonical slug	human-authority-in-company-ai-systems
Canonical path	/technical-papers/human-authority-in-company-ai-systems
Publication class	Technical architecture and implementation report
Approval state	Founder-approved public release

Revenue Cortex connection

Revenue Cortex keeps human authority explicit over consequential growth actions. The system may prepare decisions, dossiers, drafts, and recommended next actions, but pricing changes, customer-facing claims, outreach launches, CRM mutations, and delivery commitments require scoped authority records and approval gates.

Evidence-status box

> SalesFusion uses machine-readable control boundaries for human authority as a governance pattern. The pattern defines principals, roles, action classes, consequence classes, scope, conditions, expiry, approval binding, separation of duties, fail-closed behavior, and audit records. This paper describes that control pattern and the broader architecture. It does not claim measured reductions in unauthorized action, faster approvals, better decisions, or improved trust. Those claims require authorization, escalation, expiry, and adversarial-policy tests, and later field evaluation.

> The design uses governance and risk-management context consistent with ISO/IEC 42001 and the NIST AI Risk Management Framework. It uses least-trust and resource-scoped access ideas consistent with NIST SP 800-207, "Zero Trust Architecture." These references support control framing only.

Abstract

Company AI systems need human authority, but "ask a human" is not an architecture. A human instruction must be translated into a record the system can enforce: who can decide, what they can decide, which action class is covered, what consequence class is involved, what scope and conditions apply, when the authority expires, whether approval is bound to a specific action version, and which duties must remain separated. Without that structure, an AI system can treat broad permission as universal permission, reuse stale approval, collapse approval into execution, or escalate to the wrong person.

This paper defines a human-authority model for company AI systems. It specializes the approval and authority boundaries introduced by the broader evidence-gated workflow architecture in Paper 08. It separates principals, roles, policies, approval grants, execution checks, and audit events. Unknown authority fails closed. The optimizer, workflow runtime, or drafting agent may propose work, but it cannot approve its own consequential authority expansion. SalesFusion uses this as a control pattern; effectiveness remains to be tested through adversarial authorization and escalation suites.

The problem

Business AI systems operate inside existing authority structures: owners, operators, reviewers, managers, administrators, legal approvers, security approvers, and delegated contributors. A prompt saying "get approval first" does not encode those structures. It does not specify who counts as an approver, whether the approval covers this tenant, whether the action changed after approval, whether the approval expired, whether the approver has a conflict of duty, or whether the action is too consequential for single-person approval.

The problem becomes acute when AI systems can draft, recommend, schedule, mutate records, contact external parties, or modify workflow behavior. A system may ask for approval of a proposal and then execute a modified version. It may treat a prior approval as authority for a broader channel. It may infer authority from a conversation. It may allow an optimizer to remove a guardrail because the metric looked better. These are not language problems; they are governance problems.

Human authority has to be explicit enough for machines to enforce and understandable enough for humans to review.

Contributions

This paper contributes:

- A vocabulary for principals, roles, action classes, consequence classes, scope, conditions, expiry, approval binding, separation of duties, fail-closed behavior, and audit.
- A policy decision and enforcement model for AI-assisted workflows.
- A control invariant: approval authorizes an exact attempted action; it does not execute the action.
- A rule that systems may propose authority changes but cannot approve their own policy, permission, claim, or consequential workflow expansion.
- An evaluation design for authorization, escalation, expiry, replay, and adversarial-policy testing.

Core definitions

Principal is an actor that can request, approve, execute, or review work. It may be a human user, service account, workflow runtime, connector, or system component.

Role is a named authority bundle assigned to a principal within a tenant and scope. Roles should be explicit records, not inferred from job titles in prose.

Action class describes the kind of operation: read, analyze, draft, propose record, mutate internal record, contact external party, publish, spend, change permission, change workflow, or delete/retain data.

Consequence class describes the possible harm or irreversibility: read-only, internal reversible, external reversible, external consequential, financial, legal, security-sensitive, or unknown.

Scope limits authority by tenant, resource, workflow, channel, audience, amount, time window, data classification, or objective.

Conditions are predicates that must remain true for authority to apply, such as current evidence, QA passed, suppression clear, two-person approval, or rollback available.

Expiry is the time or event boundary after which authority is no longer valid.

Approval binding links an approval to an immutable action digest, workflow version, scope, approver role, conditions, and expiry.

Separation of duties is a policy constraint designed to block the same principal or component from proposing, approving, executing, and reviewing certain high-consequence changes.

Fail-closed behavior means unknown, unmatched, expired, or ambiguous authority denies or escalates rather than allowing action.

Audit is the append-only event history that lets a reviewer reconstruct who requested, approved, executed, blocked, changed, or reviewed an action.

Authority architecture

The architecture has three core services: a role and policy registry, a policy decision point, and a policy enforcement point.

Component	Responsibility
Role registry	Stores principals, roles, assignments, scopes, and expiry.
Policy registry	Stores action and consequence rules.
Policy decision point	Evaluates requested action against role, scope, conditions, and evidence.
Policy enforcement point	Blocks, allows, or requires approval immediately before action.
Approval service	Records action-bound decisions and expiry.
Audit ledger	Records policy evaluation, approval, execution, denial, and review events.

The authority record is structured:

```
{
  "policy_id": "pol_external_contact",
  "tenant_id": "tenant_ref",
  "action_class": "external_contact",
  "consequence_class": "external_consequential",
  "required_role_refs": ["role_business_owner"],
  "scope": {
    "workflow_refs": ["wf_outbound_review"],
    "channels": ["approved_external_message"],
    "data_classification": ["public", "approved_internal"]
  },
  "conditions": [
    "claim_risk_not_high",
    "suppression_clear",
    "contact_verified",
    "qa_passed"
  ],
  "default_decision": "require_approval",
  "unknown_handling": "deny_and_escalate"
}
```

Approval is a separate record:

```
{
  "approval_id": "app_001",
  "approver_principal_ref": "principal_123",
  "approver_role_ref": "role_business_owner",
  "action_digest": "sha256:...",
  "workflow_version": "wf_v9",
  "scope": {"channel": "approved_external_message"},
  "decision": "approved",
  "conditions": ["no_material_change_before_execution"],
  "expires_at": "2026-07-23T00:00:00Z"
}
```

If the action digest changes, the approval is mismatched. If the approver's role expired before decision time, the approval is invalid. If execution occurs after approval expiry, the enforcement point denies or escalates.

Approval binding and separation of duties

Approval is not a mood, preference, or chat message. It is a bounded authorization event. The binding must include:

- Exact action digest.
- Workflow and message or artifact version.
- Target resource or audience.
- Consequence class.
- Required evidence references.
- Scope limits.
- Conditions.
- Approver principal and role.
- Decision time and expiry.

Separation of duties is required for high-consequence changes. A workflow may propose an action packet. A reviewer may approve it. A runtime may execute it. A separate review may inspect the outcome. For policy, permission, or workflow guardrail changes, the optimizer may create a change proposal but cannot approve or activate it under its own identity.

This is intended to block authority escalation by optimization, subject to adversarial authorization testing. A metric-moving experiment cannot, under the defined policy, automatically remove the suppression check, widen a contact scope, change a legal claim, or grant itself connector permissions.

Implementation

SalesFusion uses structured policy and approval records for authority controls. The implementation pattern uses tenant-scoped principals, roles, policy evaluations, approval requests, approval decisions, action authorization events, execution receipts, and audit events.

The decision flow is:

```
request action
classify action_class and consequence_class
build action_digest
evaluate role, scope, conditions, expiry, and evidence
if explicit deny: block and audit
if unknown or ambiguous: deny_or_escalate and audit
if approval required: create approval request
if approval exists: validate binding, role, version, scope, conditions, expiry
authorize execution only if enforcement check passes
record action receipt or blocker
```

The policy enforcement point must run near execution time. A decision made during planning can become stale if evidence changes, approval expires, policy is revoked, or the action packet changes. The runtime cannot rely on a planning-time note that said "approved."

Authority records should be exportable and reviewable. A tenant owner should be able to inspect active policies, role assignments, pending approvals, expired approvals, denied actions, and external consequential actions. This is part of operational control, not an administrative afterthought.

Evaluation

The first evaluation should be adversarial. It should test whether the system denies or escalates when authority is missing, expired, mismatched, out of scope, self-approved, or ambiguous.

Scenarios:

Scenario	Expected result
Approval token belongs to old workflow version.	Deny as mismatched.
Approval has correct approver but expired before execution.	Deny or escalate.
Principal attempts to expand its own role.	Deny and audit.
Optimizer proposes removal of guardrail and self-approves.	Deny self-approval.
Unknown consequence class.	Treat as stricter class and escalate.
External action requested under read-only scope.	Deny.
Approval covers one channel but execution uses another.	Deny as out of scope.
Prompt asks runtime to ignore policy.	Ignore prompt, enforce policy.

Metrics include unauthorized-action rate, false-denial rate, false-approval rate, escalation precision and recall, expiry enforcement, approval replay detection, separation-of-duties violations, and audit trace completeness. Pre-installation acceptance should require zero unauthorized actions in the adversarial suite.

Safety and failure modes

The primary failure mode is false authority: the system acts as though it has permission it does not have. Causes include inferred authority, stale role assignment, broad approval reuse, version mismatch, missing consequence classification, and approval/execution collapse.

Another failure mode is authority theater: the system asks a human to approve a vague recommendation, then executes a different concrete action. This is why the action digest matters. A human approval screen must show what will happen in human-readable form and bind the decision to the machine-readable digest.

Overblocking is also a risk. A fail-closed system can block useful work if policies are unclear. The proper response is to tighten policy coverage and escalation paths, not to let unknown authority pass.

Limitations

Machine-readable authority does not ensure wise human decisions. It does not solve organizational politics, unclear ownership, poor training, or bad policy design. It can enforce only the policy and role records it has. It also adds friction to low-risk work if consequence classes are too broad.

The current SalesFusion evidence supports saying that control boundaries are implemented. It does not support trust, incident-rate, or execution-speed claims.

Practical implications

Teams should define authority before expanding autonomy. Start with consequence classes, roles, approval requirements, expiry defaults, and fail-closed behavior. Make read-only, draft, internal reversible, external consequential, financial, legal, and security-sensitive actions distinct. Bind approval to exact action versions. Keep policy evaluation and enforcement separate from model reasoning.

Review interfaces should make authority legible: who is approving, what scope applies, when it expires, what conditions remain true, and what will be blocked if anything changes.

Conclusion

Human authority in company AI systems must be represented as enforceable state. Principals, roles, action classes, consequence classes, scope, conditions, expiry, approval binding, separation of duties, fail-closed behavior, and audit are the necessary control vocabulary. SalesFusion uses this as a governance pattern and proposes adversarial authorization tests before making stronger claims. The core rule is stable: an AI system may recommend, draft, and request, but consequential authority must remain explicit, bounded, reviewable, and human-governed.

References

- National Institute of Standards and Technology, "Zero Trust Architecture," NIST SP 800-207. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf
- National Institute of Standards and Technology, "AI Risk Management Framework." https://www.nist.gov/itl/ai-risk-management-framework
- ISO, "ISO/IEC 42001:2023 - Artificial intelligence management systems." https://www.iso.org/standard/42001

Changelog

- 2026-07-17 — Version 1.0.0: founder-approved first public release; publication metadata, evidence-status box, limitations, and verified references retained.

SALESFUSION

From Signals to Buyer Intelligence

Technical architecture and implementation report

PUBLIC RELEASE · VERSION 1.0.0 · 2026-07-17

From Signals to Buyer Intelligence

Publication metadata

Field	Value
Author	SalesFusion Research
Publication date	2026-07-17
Version	1.0.0
Canonical slug	signals-to-buyer-intelligence
Canonical path	/technical-papers/signals-to-buyer-intelligence
Publication class	Technical architecture and implementation report
Approval state	Founder-approved public release

Revenue Cortex connection

Signals-to-buyer-intelligence is one of the core perception layers of Revenue Cortex. Public signals, forwarded sources, CRM changes, job posts, content, and market events become useful only when the system separates observed facts from hypotheses, route confidence, outreach readiness, and approval state.

Evidence-status box

> SalesFusion uses a buyer-intelligence workflow pattern that separates observed signals, sourced facts, inferences, route confidence, recommendations, contact identity, and action readiness. This paper defines that pattern. It does not make reply-rate, conversion, pipeline, or sales-productivity claims. Those claims require analyst-agreement, evidence-precision, and recommendation-utility studies that have not yet been published.

> The paper uses provenance concepts consistent with W3C PROV-O, AI risk framing consistent with the NIST AI Risk Management Framework, and threat awareness consistent with the OWASP Top 10 for Large Language Model Applications. These sources support terminology and risk context only.

Abstract

Buyer-intelligence systems often treat a signal as if it were already an outreach reason. A company event, hiring post, product announcement, job change, technology mention, or public statement may be relevant, but relevance is not readiness. The observation must be sourced, converted into a fact with provenance, interpreted as an inference with uncertainty, routed to a plausible business problem, checked against contact identity, assessed for claim risk, and reviewed before an external action. Skipping those steps creates generic outreach, unsupported claims, source-language copying, stale context, and misdirected contact.

We define a stateful architecture for converting signals into buyer intelligence without collapsing evidence into action. Paper 10 specializes the final action-readiness gate as a compound state machine. The architecture has seven separable objects: observed signal, sourced fact, inference, route confidence, recommendation, contact identity, and action readiness. It also includes a strict no-copy rule: source language may inform internal reasoning, but it must not be copied into outreach. SalesFusion uses the design as an internal workflow pattern, with evaluation still needed for analyst agreement and downstream utility.

The problem

Signals are noisy. They are unevenly sourced, often stale, and frequently ambiguous. A public source may say that an organization is hiring for a role, but it does not necessarily prove budget, urgency, buying authority, dissatisfaction, timing, or fit. A webpage may describe a strategic priority, but source language written for a public audience is not safe to paste into a message. A contact profile may point to a person with a similar name, but the identity may not be the right person. A route may look plausible because it matches a familiar selling motion, but the actual evidence may be too thin.

The failure pattern is a single research_complete flag. Once set, every downstream system assumes the record is usable: the account is relevant, the contact is correct, the message can cite the event, and outreach can proceed. That is an evidence error. Some records should become questions. Some should become watchlist items. Some should be blocked because the signal is copied from a source that should not be paraphrased into outreach. Some should be suppressed because the contact, company, domain, or channel is not eligible.

Buyer intelligence should therefore be an evidence product, not a scraped-text product. The useful output is not a clever paragraph. It is a dossier of typed claims, uncertainties, routing assumptions, contact checks, action constraints, and reasons to proceed or not proceed.

Contributions

This paper contributes a buyer-intelligence object model, a signal-to-action pipeline, a no-copy outreach rule, decision rules for route confidence and action readiness, and an evaluation plan. It distinguishes seven objects that are often merged:

1. Observed signal
2. Sourced fact
3. Inference

4. Route confidence
5. Recommendation
6. Contact identity
7. Action readiness

The implementation contribution is a SalesFusion pattern for recording each object separately and blocking outbound readiness until all required states pass. The evaluation contribution is a benchmark design that tests whether analysts agree with fact extraction, inference boundaries, confidence labels, route selection, and action gating.

Definitions

Observed signal is the raw detection that something may matter. It can be a public update, structured event, document change, hiring pattern, technology mention, or internal trigger. It is not yet a fact about buying need.

Sourced fact is a claim extracted from an allowed source with provenance, effective time, restriction labels, and source locator. It should preserve what is known without expanding it.

Inference is a bounded interpretation derived from one or more facts. It must be labeled as inference, not fact. For example, a hiring pattern may suggest operational expansion, but it does not prove budget or vendor interest.

Route confidence is the system's confidence that a known business route is relevant to the account or contact, based on evidence fit, recency, specificity, source quality, and missing counterevidence.

Recommendation is a proposed next step, such as collect more evidence, route to analyst review, draft an internal brief, suppress, or prepare for approval.

Contact identity is the verified linkage between person, role, organization, channel, and permitted contact method.

Action readiness is the compound state saying whether the record can support a specific action under current evidence, policy, suppression, QA, and approval conditions.

Signal-to-intelligence architecture

The architecture is a pipeline with explicit gates rather than a text-generation chain.

Layer	Input	Output	Blocking conditions
Signal capture	Raw observation	signal_record	Source not allowed, duplicate signal, tenant mismatch
Fact extraction	Signal and source	sourced_fact	Missing provenance, restricted use, stale source
Inference	Facts and route rubric	inference_record	Fact conflict, unsupported leap, low specificity
Route scoring	Inference and fit model	route_confidence	Missing route evidence, counterevidence, weak fit
Recommendation	Route state and policies	recommendation_record	No action warranted, evidence too thin
Identity check	Person and account records	contact_identity	Unverified person, wrong organization, channel unavailable
Readiness gate	All records	action_readiness	Suppression, claim risk, QA failure, approval missing

The central design rule is that later layers may reference earlier layers but cannot rewrite them. An inference cannot edit a sourced fact. A recommendation cannot silently upgrade route confidence. A contact identity check cannot make weak signal evidence strong. This keeps the intelligence dossier auditable.

The no-copy rule is a separate guardrail:

```
Source text may be quoted only inside internal evidence review where policy allows it.
Outreach drafts must not copy source language.
Outreach claims must use approved, paraphrased, source-backed assertions.
If the system cannot produce a compliant paraphrase, action readiness is blocked.
```

The rule exists because source text may carry copyright restrictions, private context, marketing language, prompt-injection text, or claims too strong for the evidence. Even when a source is public, copying its phrasing into outreach can imply more familiarity, permission, or certainty than the workflow has.

Data objects and decision rules

A minimal buyer-intelligence record can be represented as a graph:

```
{
  "signal_record": {
    "signal_id": "sig_001",
    "source_ref": "src_public_version_7",
    "observed_at": "2026-07-16T12:00:00Z",
    "signal_type": "hiring_pattern",
    "raw_use_policy": "internal_evidence_only"
  },
  "sourced_fact": {
    "fact_id": "fact_001",
    "signal_ref": "sig_001",
    "statement": "The source indicates an open role in a relevant function.",
    "effective_at": "2026-07-16T12:00:00Z",
    "epistemic_status": "observed_fact",
    "lifecycle_state": "active",
    "restrictions": ["no_source_language_in_outreach"]
  },
  "inference_record": {
    "inference_id": "inf_001",
    "fact_refs": ["fact_001"],
    "inference": "The organization may be expanding a function related to the route.",
    "confidence": "medium",
    "missing_evidence": ["budget", "owner", "active project"]
  }
}
```

Route confidence is a separate scorecard, not a free-form judgment. A conservative rubric uses evidence dimensions:

Dimension	Question	Example labels
Recency	Is the signal current enough for the route?	current, aging, stale
Specificity	Does the fact identify a concrete business condition?	direct, adjacent, generic
Source quality	Is the source allowed and traceable?	primary, secondary, unknown
Route fit	Does the route map to the observed condition?	strong, plausible, weak
Counterevidence	Is there evidence against the route?	none found, unresolved, blocking
Contact fit	Is the person plausibly connected to the route?	verified, possible, unverified

The recommendation rule should prefer restraint:

```
if source is restricted or stale:
  recommend collect_evidence
elif inference has unresolved conflict:
  recommend analyst_review
elif route_confidence is weak:
  recommend watchlist_or_no_action
elif contact_identity is unverified:
  recommend identity_research
elif suppression applies:
  recommend suppress
else:
  recommend prepare_action_packet
```

Preparing an action packet is not the same as sending. It only means the record may proceed to readiness, QA, and approval checks.

Implementation

SalesFusion uses this pattern as a structured workflow rather than a single research prompt. Signals are recorded with source references and observed time. Facts are extracted into constrained fields with provenance and restrictions. Inferences are labeled, tied to fact references, and prevented from becoming sourced facts. Route confidence is calculated from a rubric that includes recency, specificity, source quality, route fit, counterevidence, and contact fit. Contact identity is handled separately from account intelligence.

The implementation stores null and blocked states. For example, no_contact_verified, source_restricted, route_confidence_low, claim_risk_high, and suppression_match are valid outcomes of research. They are not treated as failures to be hidden. They provide explicit blocker states when a signal does not support an outreach action.

Interfaces should expose the distinction in review screens. A reviewer should see the observed signal, sourced facts, inferences, confidence rationale, recommendation, identity status, and readiness blockers as different fields. Generated copy, if produced, must be downstream of those fields and cannot directly quote the source. A reviewer should be able to reject the copy while keeping the intelligence record, or reject the inference while preserving the fact.

Evaluation

The first evaluation should measure whether the intelligence objects are correct and useful to reviewers, not whether revenue changes. A benchmark can provide public-safe or synthetic signal packages with ground-truth labels for source eligibility, fact extraction, inference boundary, route fit, contact identity, and readiness state.

Suggested metrics:

Metric	Purpose
Evidence precision	Extracted facts are supported by allowed source material.
Unsupported-inference rate	Inferences exceed the facts or are mislabeled as facts.
Analyst agreement	Independent reviewers agree on route confidence and action readiness.
Source-copy violation rate	Outreach drafts contain prohibited copied source language.
Identity error rate	Recommended contact is not verified for the organization and route.
False-ready rate	Record is marked ready despite suppression, identity, source, or claim-risk blockers.
Recommendation utility	Reviewers judge recommended next step as appropriate under the evidence.

The evaluation should preserve negative labels. A good system may produce many collect_more_evidence, watchlist, or no_action records. That is not necessarily poor performance; it may be correct restraint under weak signals.

Risks and failure modes

The main risk is laundering. A generated dossier can make weak evidence look specific. It can turn a public fact into a private-sounding claim, imply urgency where none exists, or describe an inferred business problem as if the buyer confirmed it. Another risk is identity drift: a signal about an organization is routed to a person whose role, employment, or contact channel is not verified. Source prompt injection is also relevant: a source can contain text that attempts to instruct the agent to ignore rules or contact someone.

Controls include source allowlists, provenance recording, fact/inference separation, confidence rubrics, no-copy enforcement, identity verification, suppression checks, QA, and approval gates. OWASP LLM application risks are useful in reviewing these controls because buyer-intelligence workflows combine retrieval, generated text, external tools, and sensitive business context.

Limitations

The architecture cannot infer buyer intent from thin evidence. It cannot ensure that public signals are accurate, current, or meaningful. It cannot prove that a recommended route is commercially valuable. The SalesFusion pattern supports evidence and confidence separation, but the effect on analyst speed, message quality, or sales outcomes remains unevaluated.

There is also a judgment cost. Separating objects creates more fields to review. Some teams may prefer faster but less governed research for low-consequence internal tasks. The full pattern is better suited where external outreach, reputation, compliance, or data-use restrictions matter.

Practical implications

Teams should treat buyer intelligence as a state record. A useful record says what was observed, what fact was extracted, what inference was made, why a route is plausible, who the contact is, what action is recommended, and why action is or is not ready. It should be normal for a signal to end in no_action.

The no-copy rule should be enforced at the system level, not left as a style preference. Generated outreach should be based on approved claims and constraints, not pasted source language. Reviewers should approve a claim packet and an action packet, not a scraped paragraph.

Conclusion

Signals are not buyer intelligence until they are sourced, interpreted, routed, checked, and gated. SalesFusion uses a workflow pattern that keeps observed signals, sourced facts, inferences, route confidence, recommendations, contact identity, and action readiness separate. The pattern is designed to block or flag unsupported outreach and source-language copying while preserving useful evidence for human review. Its

practical value should be tested through analyst agreement, evidence precision, recommendation utility, and false-ready benchmarks before making business-outcome claims.

References

- W3C, "PROV-O: The PROV Ontology." https://www.w3.org/TR/prov-o/
- National Institute of Standards and Technology, "AI Risk Management Framework." https://www.nist.gov/itl/ai-risk-management-framework
- OWASP Foundation, "OWASP Top 10 for Large Language Model Applications." https://owasp.org/www-project-top-10-for-large-language-model-applications/

Changelog

- 2026-07-17 — Version 1.0.0: founder-approved first public release; publication metadata, evidence-status box, limitations, and verified references retained.

SALESFUSION

Installing an Outcome-Oriented Revenue Operating System

Technical architecture and implementation report

PUBLIC RELEASE · VERSION 1.0.0 · 2026-07-17

Installing an Outcome-Oriented Revenue Operating System

Publication metadata

Field	Value
Author	SalesFusion Research
Publication date	2026-07-17
Version	1.0.0
Canonical slug	installing-outcome-oriented-revenue-os
Canonical path	/technical-papers/installing-outcome-oriented-revenue-os
Publication class	Technical architecture and implementation report
Approval state	Founder-approved public release

Revenue Cortex connection

This paper describes the installation method behind SalesFusion's Revenue Cortex: a governed revenue operating layer installed around a company's product and go-to-market motion. Revenue Cortex extends the Revenue OS idea with source-backed memory, attention rules, executive authority, first running jobs, proof-of-run, outcome ledgers, and continual operating review.

Evidence-status box

> This paper defines an installation method and describes the SalesFusion delivery design. The method begins with governed Company Memory, explicit authority, one First Running Job, evidence-producing reporting, operator control, and an optimization cadence. It has not been evaluated as a multi-installation field study, and it does not claim revenue improvement, conversion lift, efficiency gain, or client outcome effects. The allowed claim is method-level: an agentic revenue system can be installed through governed context, one working motion, evidence, reporting, and ongoing optimization cadence.

Abstract

An outcome-oriented Revenue OS should not begin with broad automation. It should begin by making company state explicit, choosing one narrow operating motion, testing a non-mutating or tightly controlled job, and reporting evidence before expanding scope. We define an installation method for an agentic Revenue OS

built on Company Cortex principles. Paper 07 specializes this installation method into the First Running Job activation protocol. The sequence moves from readiness and authority assessment to Company Memory, objective tree, approval paths, First Running Job, observability, live gate, reporting, operator handoff, rollback drill, and annual optimization cadence. The paper specifies data objects, states, interfaces, acceptance criteria, safety controls, evaluation design, and failure modes. It avoids commercial terms and does not describe organization-specific workflows. The method is intended for practitioners who need a repeatable way to install governed revenue operations without confusing activity, autonomy, or dashboard output with verified outcome evidence.

The problem

Revenue automation projects often start in the wrong place. A team connects systems, generates copy, scores accounts, or launches workflows before it has a reliable representation of objectives, constraints, sources, authority, suppression rules, operating state, and proof of completion. The result may look active, but activity is not an operating system. Without canonical memory, the system cannot explain what it believes. Without authority, it cannot know what it may do. Without run evidence, it cannot prove that work completed. Without outcome definitions, it cannot distinguish useful motion from local proxy activity.

The installation problem is therefore a sequencing problem. A governed Revenue OS should order the work around ambiguity: state before recommendation, authority before action, evidence before expansion, reporting before optimization, and rollback before broader autonomy.

Contributions

This paper contributes a repeatable installation method.

First, it defines a thirty-day sequence from Company Memory through one running job and reporting.

Second, it defines installation artifacts: readiness assessment, source matrix, objective tree, authority map, Activation Card, run evidence contract, reporting ledger, and optimization backlog.

Third, it specifies operator control: inspect, pause, reject, correct, approve, export, and roll back.

Fourth, it defines the annual operating cadence without public commercial terms or unsupported outcome claims.

Fifth, it states evaluation requirements for future field evidence.

Core definitions

Revenue OS means a governed operating layer for revenue workflows, decisions, evidence, and optimization. In this paper, it is an installation method, not a claim of revenue impact.

Outcome-oriented means objectives, metrics, and guardrails are represented before workflow expansion. It does not mean outcomes are promised.

Company Memory baseline is the initial source-backed state snapshot for relevant objectives, constraints, entities, policies, and unknowns.

First Running Job is the first narrow workflow selected for proof. It should be non-mutating or tightly bounded before live activation.

Activation Card is the pre-run contract for objective, scope, evidence, authority, simulation, live gate, rollback, and expansion criteria.

Operator control is the human ability to inspect, pause, reject, correct, approve, export, and roll back system behavior.

Annual optimization cadence is the weekly, monthly, and quarterly review structure for run health, constraints, experiments, policies, and regression.

Installation model

The method has six phases.

Phase	Purpose	Required artifact
readiness	define scope, owners, sources, authority, connectors, risk	readiness and authority assessment
memory	build source-backed current state	Company Memory snapshot
control	define objectives, constraints, approvals, consequence classes	objective and authority map
first job	design one narrow workflow and evidence contract	Activation Card
proof and live gate	simulate, verify, approve, and run under bounded scope	proof run and live decision
operate	report, correct, optimize, and monitor regression	reporting ledger and backlog

The installation does not require every possible workflow to be modeled before value can be inspected. It does require every consequential action in the first workflow to have state, authority, and evidence boundaries.

Thirty-day implementation sequence

The sequence below is a public-safe method, not a promise of timing or outcome.

Window	Work	Evidence
Pre-install	readiness, scope, owner, objective, authority, source, connector assessment	signed scope, source/access matrix, first-job hypothesis
Days 1-7	tenant kernel, ontology, source registry, Company Memory baseline, objective tree	source-backed snapshot, conflicts, unknowns, restrictions
Days 8-14	policies, consequence classes, approval paths, first workflow design	tested approval and denial cases, Activation Card
Days 15-21	adapters, non-mutating proof, event capture, observability	dated proof runs, blockers, evidence, no unintended mutation
Days 22-30	controlled activation, reporting, training, rollback drill	one live job with evidence, operator signoff, rollback proof

The source/access matrix should list source owner, classification, effective time, review cadence, allowed uses, restrictions, and connector permission. The objective tree should define primary outcome, subgoals, metrics, guardrails, anti-goals, and tradeoffs. The authority map should define who can approve read-only, internal reversible, external reversible, external consequential, financial, legal, and security-sensitive actions.

From Company Memory to one running job

The installation begins with Company Memory because a Revenue OS cannot operate from ungoverned fragments. The first snapshot should include relevant objectives, current constraints, key entities, active policies, known conflicts, unknowns, stale records, and restricted sources. Every current-state field should trace to a source or be labeled unknown.

The first workflow is chosen after memory and authority are visible. Selection criteria:

Criterion	Reason
narrow scope	reduce blast radius
known failure cases	make proof meaningful
evidence-producing	verify completion beyond process exit
non-mutating proof possible	test before external consequence
objective relevance	avoid activity-only automation
operator review feasible	enable correction and handoff

The First Running Job should first run in simulation or non-mutating proof mode. If it produces an artifact, the expected artifact schema is defined before the run. If it touches an external system later, idempotency, approval, action receipt, and rollback are required.

Reporting and operator control

Reporting should be deterministic where possible. AI can explain records, but it should not silently alter totals, invent missing values, or change metric definitions. A reporting ledger should include run status, verified completion, blockers, costs, approvals, artifacts, outcome observations, correction records, and open risks.

Operator controls must be present before broader activation:

Control	Required behavior
inspect	view sources, decisions, runs, approvals, artifacts
pause	stop workflow from a known checkpoint
reject	decline recommendation or approval request
correct	submit memory correction through validation
approve	authorize exact action, version, scope, expiry
export	retrieve sources, state, policies, decisions, artifacts
roll back	return to prior version where feasible and verify

OpenTelemetry concepts can support infrastructure observability, but operating evidence needs domain events: decision.recommended, approval.approved, run.output_created, run.verified_complete, run.rollback_started, and action.receipt_recorded.

Implementation

SalesFusion has implemented this as a delivery design: Company Memory first, one First Running Job, reporting, operator control, and ongoing review cadence. A reference implementation can use the Company Cortex service families:

```
/sources
/assertions
/state/snapshot
/objectives
/policies
/approvals
/decisions
/workflows
/runs
/experiments
/outcomes
/exports
```

Implementation invariants:

Invariant	Installation check
no source-free current state	every active field traces to evidence or unknown
no unknown consequential authority	deny or escalate unmatched action
no approval/execution collapse	approval event does not call connector
no false completion	expected artifact and receipt contract required
no optimizer self-approval	experiments and changes require separate authority
no tenant ambiguity	every object, index, artifact, and export is scoped

The technology stack should be selected after these contracts are stable. Relational constraints are useful for canonical records, append-only events for reconstruction, object storage for source artifacts, search indexes for disposable projections, and queue/idempotency patterns for workflow reliability.

Evaluation

Installation evaluation has three levels.

Contract evaluation checks schema validity, state-machine legality, provenance fields, approval binding, idempotency, tenant isolation, export integrity, and rollback events.

Scenario evaluation checks stale facts, contradictory sources, missing authority, expired approval, prompt injection, duplicate events, failed connectors, misleading proxy improvement, objective conflict, irreversible action request, and failed experiment.

Field evaluation should come only after controlled installations. It would need method, sample, baseline, time window, confounders, null and negative results, and limitations. Measures might include operator correction burden, time to trustworthy state reconstruction, recommendation acceptance and rejection reasons, false-ready rates, unauthorized-action rates, run reliability, rollback success, decision cycle time, and outcome movement with confounders recorded. This paper does not report those measures.

Safety, risks, and failure modes

Installation risks include connecting too much too early, treating a source upload as memory, granting broad authority, launching a mutating workflow before proof, accepting process exit as completion, optimizing activity metrics, exposing restricted data in reports, and expanding scope before rollback is tested. OWASP LLM application risks are relevant to prompt injection and tool misuse; NIST SP 800-207 supports resource-scoped access reasoning; NIST AI RMF supports risk-management framing. None of these references validate the installation method's outcomes.

The method targets these risks by sequencing evidence before expansion. It can still fail if humans approve inappropriate scope, sources are poor, connectors drift, or objectives are ambiguous.

Limitations

The method is operationally demanding. It requires owners, source access, authority clarity, review time, and willingness to expose unknowns. Some teams may prefer faster automation with fewer controls; this method prioritizes inspectability over speed. It also does not solve weak strategy, poor data quality, or unclear ownership by itself.

The thirty-day sequence is a design sequence, not a promise that every organization can reach the same state in the same time. Complex environments may need longer readiness, security, source cleanup, or authority work.

Practical implications

A team installing an agentic Revenue OS should resist broad workflow launch until four things exist: source-backed current state, machine-readable authority, one proof workflow, and reporting that distinguishes run activity from verified evidence. Expansion should be based on inspected proof, not enthusiasm for automation.

Annual operation should follow a cadence. Weekly reviews inspect runs, blockers, approvals, and corrections. Monthly reviews reassess constraints, experiments, proxy risks, and version changes. Quarterly reviews examine objectives, permissions, connectors, operating cost, data quality, exports, and workflow retirement.

References used

This paper uses NIST AI Risk Management Framework for risk framing, NIST SP 800-207 for resource-scoped access reasoning, OpenTelemetry Specification for observability concepts, and OWASP Top 10 for Large Language Model Applications for threat awareness. These references do not validate SalesFusion-specific installation outcomes.

Conclusion

An outcome-oriented Revenue OS should be installed as a governed operating system, not launched as broad automation. The sequence is Company Memory, authority, one First Running Job, proof, reporting, operator control, rollback, and optimization cadence. SalesFusion has implemented the delivery design, while field effectiveness remains unevaluated. The practical standard is one running job with evidence before wider scope.

Changelog

- 2026-07-17 — Version 1.0.0: founder-approved first public release; publication metadata, evidence-status box, limitations, and verified references retained.

SALESFUSION

The Consulting Portal as an Implementation Workspace

Technical architecture and implementation report

PUBLIC RELEASE · VERSION 1.0.0 · 2026-07-17

The Consulting Portal as an Implementation Workspace

Publication metadata

Field	Value
Author	SalesFusion Research
Publication date	2026-07-17
Version	1.0.0
Canonical slug	consulting-portal-implementation-workspace
Canonical path	/technical-papers/consulting-portal-implementation-workspace
Publication class	Technical architecture and implementation report
Approval state	Founder-approved public release

Revenue Cortex connection

The consulting portal is one workspace through which a Revenue Cortex can be installed and reviewed. It should preserve canonical inputs, client decisions, SalesFusion outputs, approvals, revisions, deliverables, and evidence without pretending that a portal alone is the revenue system.

Evidence-status box

> SalesFusion uses a governed workspace foundation for consulting implementation work. The foundation separates curriculum, canonical state, submissions, reviews, versions, decisions, deliverables, evidence, outcomes, tenant isolation, and AI draft boundaries. This paper describes that foundation and proposes the full workspace model. It does not claim measured improvements in client completion, review-cycle time, information retrieval, satisfaction, or project outcomes. Those claims require client usability, completion, review-cycle, and retrieval studies.

> The design uses governance context consistent with ISO/IEC 42001, resource-scoped access principles consistent with NIST SP 800-207, and HTTP interface semantics consistent with RFC 9110. These references support architecture framing only.

Abstract

Complex AI consulting is often run through calls, documents, chat threads, task lists, and shared folders. That structure is workable for small engagements but weak for governed implementation. It scatters state, loses review boundaries, mixes drafts with decisions, hides evidence, and makes it difficult to know which version of a deliverable is current. A chatbot transcript is not enough. A governed client implementation workspace needs canonical state, submissions, reviews, decisions, versions, deliverables, evidence, outcome records, and tenant isolation.

This paper defines the consulting portal as an implementation workspace rather than a content library or chat interface. The workspace organizes curriculum, client inputs, AI-generated drafts, consultant review, approved decisions, deliverables, and outcome records into typed state. AI can draft, summarize, identify missing evidence, and propose next steps. It cannot create final approval, silently modify canonical state, or treat a draft as a client decision. SalesFusion uses the workspace foundation; full usability and outcome evidence remain pending.

The problem

Consulting implementation work has an information-shape problem. A client answers questions in one place, uploads context in another, reviews a draft in a third, and approves a decision verbally or by message. Later, the team needs to know what the client actually approved, which evidence supported it, which deliverable version is final, which assumptions remain open, and whether the implementation produced the expected artifact. Without a governed workspace, those answers require manual reconstruction.

AI can make this worse. A model can summarize a transcript as if it were canonical state. It can turn a tentative answer into a final decision. It can merge old and new information without supersession. It can generate a polished deliverable that looks approved because it is well formatted. It can answer from another tenant's pattern if isolation is weak. In consulting work, the risk is not only data leakage. It is decision confusion: draft, review, approval, and delivery collapse into one persuasive document.

The portal should therefore be a stateful implementation environment. Its job is to help the client and operator move from curriculum and inputs to reviewed decisions and deliverables with explicit evidence and ownership.

Contributions

This paper contributes:

- A reference model for a governed consulting implementation workspace.
- Definitions for curriculum, canonical state, submissions, reviews, versions, decisions, deliverables, evidence, outcomes, and tenant isolation.
- A rule that AI draft is not final approval.
- A workflow model for submission, analysis, review, decision, deliverable generation, outcome recording, and version supersession.
- Implementation invariants for tenant scope, state ownership, review gates, and exportability.

- An evaluation design for usability, completion, review-cycle behavior, and retrieval accuracy.

Core definitions

Workspace is a tenant-scoped implementation environment where curriculum, client inputs, reviews, decisions, deliverables, and outcomes are structured records.

Curriculum is the ordered implementation content: modules, prompts, exercises, required inputs, acceptance criteria, and dependencies.

Canonical state is the current structured representation of the client's implementation context, built from approved submissions, reviewed evidence, and decisions. It is not a chat summary.

Submission is a client or operator input awaiting validation, review, or incorporation.

Review is a structured assessment of a submission, draft, deliverable, evidence packet, or decision request.

Version is an immutable or superseded record of a state object, deliverable, curriculum item, or decision packet.

Decision is an approved, rejected, deferred, or superseded choice with authority, evidence, scope, and effective time.

Deliverable is a versioned artifact created for client use, such as a plan, map, checklist, report, configuration, or implementation packet.

Evidence is a source, submission, artifact, review, or event that supports a state field, decision, or deliverable.

Outcome is a recorded implementation result, including completed, blocked, failed, no-result, superseded, or pending states.

Tenant isolation means each client workspace has scoped identity, storage, retrieval, permissions, exports, and audit boundaries.

AI draft is generated content requiring review. It is not canonical state, a final deliverable, or an approved decision until accepted through the appropriate gate.

Workspace architecture

The workspace has seven layers:

Layer	Purpose	Example records
Curriculum	Guides implementation sequence	module, task, prerequisite, acceptance criterion
State	Stores current approved context	canonical field, unknown, conflict, supersession
Submission	Captures human input	answer, upload, comment, correction
Review	Evaluates inputs and drafts	review note, issue, approval request
Decision	Records authority	decision packet, approval, rejection, deferral
Deliverable	Produces client artifacts	artifact version, export, final packet
Outcome	Tracks implementation result	completion, blocker, rollback, learning record

The workflow is not linear in practice, but state transitions should be explicit:

```
curriculum.assigned
submission.created
submission.validated
ai_draft.created
review.requested
review.completed
decision.requested
decision.approved | decision.rejected | decision.deferred
canonical_state.updated
deliverable.version_created
deliverable.approved
outcome.recorded
```

AI draft is deliberately before review and decision. A generated implementation plan can be useful, but it cannot be treated as final merely because it exists. A draft must reference source submissions and current canonical state. If the draft includes unsupported claims or fills unknowns, review should block it.

Data model and invariants

A submission record:

```
{
  "submission_id": "sub_001",
  "tenant_id": "tenant_ref",
  "curriculum_item_ref": "module_03_question_05",
  "submitted_by": "principal_ref",
  "submitted_at": "2026-07-16T14:00:00Z",
  "content_type": "structured_answer",
  "state": "awaiting_review",
  "evidence_refs": [],
  "supersedes": null
}
```

A decision record:

```
{
  "decision_id": "dec_001",
  "tenant_id": "tenant_ref",
  "decision_type": "implementation_scope",
  "subject_refs": ["state_field_12", "deliverable_v4"],
  "evidence_refs": ["sub_001", "review_002"],
  "decision": "approved",
  "approved_by_role": "client_owner",
  "effective_at": "2026-07-16T15:00:00Z",
  "expires_at": null,
  "conditions": ["applies_to_current_version_only"]
}
```

Core invariants:

Invariant	Purpose
A canonical state field must trace to approved evidence or an explicit unknown.	Guards against chat summaries becoming facts.
A submission cannot silently overwrite canonical state.	Preserves review and supersession.
An AI draft cannot be final approval.	Preserves human authority.
A deliverable version must identify source state, evidence, reviewer, and approval state.	Keeps version relationships reviewable.
Tenant identifier is required on every object and query.	Supports isolation and export.
Decisions are effective-time records and can be superseded, not erased.	Preserves auditability.

Implementation

SalesFusion uses the workspace foundation as a portal model with structured curriculum, submissions, reviews, versioned deliverables, evidence references, and state boundaries. The implementation pattern is intentionally different from a chat transcript. Chat or AI assistance may exist inside the workspace, but it writes proposals, drafts, or review aids. Canonical state changes require validation and approval events.

The interface should make state visible without turning the portal into administrative clutter. A user should see assigned curriculum, pending submissions, unresolved questions, drafts needing review, decisions awaiting approval, current deliverables, and blockers. The system should preserve prior versions and show what changed between versions.

Tenant isolation is enforced at the resource level. Each request is authenticated and tenant-scoped. Cross-tenant object identifiers should fail closed and create audit events. Exports should include only tenant-scoped, permitted, non-secret material with a manifest of included objects.

An implementation API can use resource boundaries such as:

```
POST /workspaces
GET  /workspaces/{id}/curriculum
POST /submissions
POST /submissions/{id}/review
POST /drafts
POST /decisions
POST /deliverables
POST /deliverables/{id}/versions
POST /outcomes
POST /exports
```

HTTP success indicates the request was accepted by the service, not that consulting work is complete. Completion requires the relevant review, decision, deliverable, and outcome state.

Evaluation

The evaluation should begin with workflow correctness and user comprehension. It should not start with claims about project outcomes. A realistic study can ask client-side and operator-side participants to complete tasks inside a controlled workspace.

Tasks:

- Find the source behind a canonical state field.
- Correct a stale submission.
- Distinguish AI draft from approved deliverable.
- Approve one decision and reject a broader one.
- Locate the current version of a deliverable.
- Identify unresolved blockers.
- Export workspace state.
- Reconstruct why a decision was made.

Metrics:

Metric	Definition
Task completion	Users can complete required workspace tasks.
Review-cycle clarity	Users can identify what needs review and why.
Draft/approval distinction	Users do not mistake AI drafts for final decisions.
Retrieval accuracy	Users find current state and evidence without wrong-version errors.
Correction success	Stale or incorrect submissions are corrected without overwriting history.
Tenant isolation tests	Cross-tenant access attempts fail closed.
Export completeness	Export contains expected permitted objects and excludes restricted material.

Later field studies can measure completion, review-cycle duration, support requests, and implementation outcomes, but only with visible method, sample, baseline, and limitations.

Risks and failure modes

The main failure mode is state confusion. A draft becomes treated as final. A submission becomes treated as approved. A prior deliverable version remains in circulation. A decision is remembered but not recorded. Another failure is tenant leakage, where retrieval, examples, indexes, exports, or operator views cross workspace boundaries.

AI-specific failures include unsupported completion of missing fields, overconfident summaries, accidental disclosure of restricted content, and prompt injection inside uploaded material. Controls include draft labeling, review gates, provenance, tenant-scoped indexes, source handling rules, export checks, and audit events.

There is also a product risk: too much governance can make the workspace feel heavy. The design should expose the state needed for action without forcing users to manage implementation metadata directly.

Limitations

The workspace model does not ensure successful consulting delivery. It cannot make poor curriculum good, replace expert review, or force client participation. Tenant isolation depends on correct implementation and testing. AI draft boundaries depend on product surfaces that make draft status unmistakable.

SalesFusion's current evidence supports an implemented workspace foundation. It does not support claims about usability improvement, faster completion, reduced project risk, or better outcomes.

Practical implications

Consulting portals should be designed around implementation state, not around content browsing. Every important client answer, review, decision, deliverable, and outcome should have a record, version, evidence link, and owner. AI should accelerate drafting and review preparation, but final authority should remain a separate state transition.

Teams should invest early in tenant isolation, versioning, export, and draft labeling. These controls are harder to retrofit after users rely on the workspace as the system of record.

Conclusion

A consulting portal for AI implementation should be a governed workspace. Curriculum, canonical state, submissions, reviews, versions, decisions, deliverables, evidence, outcomes, tenant isolation, and AI draft boundaries are the structure intended to keep implementation work reviewable rather than scattered across transcripts and ambiguous documents. SalesFusion uses the foundation of this model and proposes usability and workflow evaluations before making stronger claims.

References

- ISO, "ISO/IEC 42001:2023 - Artificial intelligence management systems." https://www.iso.org/standard/42001
- National Institute of Standards and Technology, "Zero Trust Architecture," NIST SP 800-207. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf
- IETF, "RFC 9110: HTTP Semantics." https://www.rfc-editor.org/rfc/rfc9110.html

Changelog

- 2026-07-17 — Version 1.0.0: founder-approved first public release; publication metadata, evidence-status box, limitations, and verified references retained.