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.