

Isotopes AI | aidnn

The Orchestration Imperative for AI Agents

Principles from Organizational Design for Agentic Systems

Version 1.0 | January 2026



Executive Summary	4
Introduction	5
Structured Agent Teams	6
Discovery Team	6
Strategy Team	6
Operations Team	7
Verification Team	7
Specialist Team	7
Information Flow	8
Why Multi-Agent Systems Fail in Practice	9
Coordination Overhead Scales Nonlinearly	9
Context Contamination and Goal Drift	9
Debugging and Trust Collapse	9
The Seven Principles Behind aidnn	9
1. Structure Over Freedom	9
Design implications	10
Common anti-patterns	10
Adoption checklist	10
2. Hierarchy Tames Complexity	11
Design implications	11
Common anti-patterns	11
Adoption checklist	11
3. Plans Are Contracts	12
Design implications	12
Common anti-patterns	12
Adoption checklist	13
4. Leverage Through Information	13
Design implications	13
Common anti-patterns	14
Adoption checklist	14
5. Iterate Tactics, Not Strategy	14
Design implications	15
Common anti-patterns	15
Adoption checklist	15
6. Emergence Over Programming	15
Design implications	16
Common anti-patterns	16
Adoption checklist	16
7. Humans Own the What	16
Design implications	17
Common anti-patterns	17
Adoption checklist	17
Putting the Principles to Work in Enterprise Analytics	17
Reliability and Reproducibility	17
Governance and Trust	17
Cost and Performance	18
Organizational Learning	18
Key Takeaways: What to Evaluate in an Agentic Systems	18
Conclusion	19



Executive Summary

Enterprises are rapidly adopting Agentic AI Systems powered by Large Language Models (proprietary and open) to automate workflows across use cases like content generation & management, customer support, insurance underwriting, and many more! An obvious emerging use case is the need to democratize access to data and data analysis to accelerate decision-making. Yet most “chat-with-your-data” experiences break down when questions become ambiguous, high-stakes, or cross-system: definitions differ across teams, the relevant data is scattered, and correctness requires a chain of reasoning that must be transparent and reviewable.

aidnn is an agentic data analyst built around a simple thesis: analytics is not a chatbot problem - it is an orchestration problem. The core challenges of orchestrating complex work have been studied for decades in software engineering and management. aidnn applies that wisdom to agentic AI by emphasizing structured workflows, role clarity, planning as a contract, information leverage, tactical iteration within strategic constraints, and human oversight. The result: overall turnaround drops from days or hours to minutes, and every follow-up question that once required another analyst cycle now resolves in minutes.

This whitepaper distills the principles behind aidnn and shows how each principle translates into reliable system behavior in production. It is intended for enterprise executives, product users, technical evaluators, and investors seeking a clear framework for what “production-grade agentic AI” requires.

THE SEVEN PRINCIPLES AT A GLANCE

1. **Structure Over Freedom:** Defined stages, not open-ended chat
2. **Hierarchy Tames Complexity:** Coordination through role clarity
3. **Plans Are Contracts:** Explicit specs before execution
4. **Leverage Through Information:** Right context at the right time
5. **Iterate Tactics, Not Strategy:** Refine execution, protect intent
6. **Emergence Over Programming:** Simple components, sophisticated behavior
7. **Humans Own the What:** Human judgment shapes intent, system handles how



Introduction

A question like “What’s our ARR by product?” sounds straightforward, but the work behind it rarely is. The data is distributed across warehouses, CRM systems, billing platforms, and spreadsheets. Business definitions diverge across teams. Revenue must be normalized across billing frequencies, proration rules, and edge cases. And the answer must be presented in a way that decision-makers can trust.

In early prototypes, it is tempting to treat these challenges as prompt engineering problems: expand context, add instructions, and hope the model behaves. In production, this approach collapses under scale and variability. Token budgets, shifting schemas, ambiguous definitions, and high-stakes accountability demand a system that is structured, observable, and correct-by-design.

aidnn’s architecture is motivated by the observation that effective AI agent teams resemble effective human teams. Communication overhead, role specialization, quality control, and explicit specifications matter just as much in AI workflows as they do in engineering organizations. The rest of this paper describes seven principles that guide aidnn’s design and how those principles translate to production-grade outcomes.

aidnn approaches analytics as a structured workflow with five distinct phases: Ask, Plan, Refine, Execute, and Operationalize. Users begin by articulating their question in natural language. The system then builds a reviewable plan, surfacing assumptions, definitions, and data sources before any execution occurs. The Refine phase, central to the workflow, enables iterative improvement through human feedback and system critique. Execution proceeds only after alignment. Finally, validated analyses can be operationalized into reusable assets, turning one-off questions into organizational knowledge.

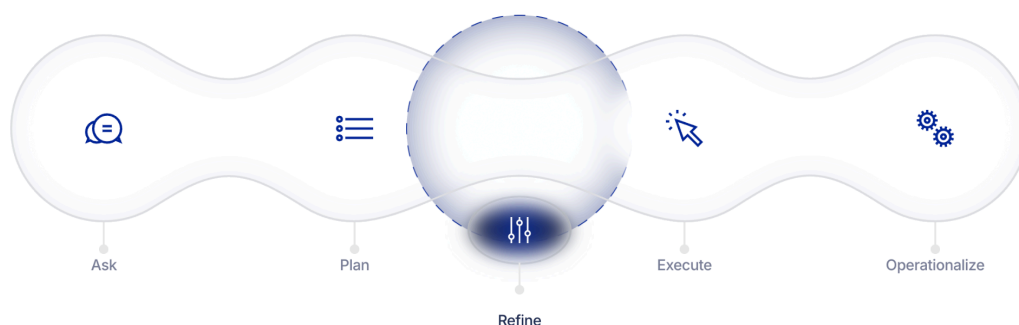


Figure 1: The aidnn workflow. Refinement sits at the center, ensuring human alignment before costly execution.



Structured Agent Teams

Organizations do not achieve complex objectives through unstructured collaboration. They form teams with distinct mandates, clear interfaces, and defined handoffs. A product launch involves strategy teams setting direction, engineering teams building, quality teams validating, and operations teams deploying. Each team has specialized skills, receives what it needs from upstream, and delivers defined outputs downstream.

aidnn mirrors this structure. Rather than a single agent attempting everything, specialized teams handle distinct phases of the analytical workflow. Information flows top-down from strategic planning to tactical execution, with each team operating within constraints set by the team above.

Discovery Team

Analogous to Business Analysts, Data Engineers, and Requirements Analysts

Before planning can begin, the Discovery Team gathers the information necessary for informed decision-making. This includes retrieving relevant database schemas through progressive metadata discovery, surfacing prior insights from organizational memory, and consulting domain experts for methodology guidance.

The Discovery Team operates on a principle of progressive disclosure: broad semantic search first, then precision filtering, then detailed retrieval. This staged approach manages token budgets while ensuring planners receive complete, relevant context.

Strategy Team

Analogous to VPs, Solution Architects, and Technical Program Managers

The Strategy Team functions as aidnn's planning layer. It receives the user's question along with assembled context (relevant schemas, organizational memory, domain expertise) and produces a formal specification: the plan. This plan defines what will be accomplished, which agents will execute each step, what data flows between steps, and what assumptions underlie the approach.

The Strategy Team does not execute. Its output is a contract that downstream teams fulfill. This separation ensures that strategic decisions are made with full context, not improvised during implementation.



Operations Team

Analogous to Engineering Managers, Senior Engineers, and Development Teams

The Operations Team implements the plan exactly as specified. Specialized agents (code writers, data transformers, visualization creators) each handle steps matching their capabilities. Operations agents see only their assigned step and the outputs from prior steps. They do not see planning discussions or other agents' work.

This constrained visibility is deliberate. Operations agents fulfill the contract without renegotiating it. When implementation reveals issues, they escalate rather than improvise architectural changes.

Verification Team

Analogous to QA Engineers, Code Reviewers, and Test Engineers

The Verification Team provides critique and quality assurance throughout execution. After each step, critique agents evaluate outputs against specifications: Does the code handle edge cases? Does the transformation preserve data integrity? Does the visualization accurately represent the underlying data?

Verification operates tactically. It improves how steps are accomplished without changing what is being accomplished. When verification consistently fails, the system escalates to replanning rather than accumulating tactical patches.

Specialist Team

Analogous to Principal Engineers, Domain Consultants, and Subject Matter Experts

The Specialist Team provides specialized knowledge that general-purpose agents lack. Financial experts contribute revenue recognition methodology. Reconciliation experts provide matching algorithms and tolerance thresholds. Visualization experts recommend appropriate chart types for different data patterns.

Specialists are consulted selectively, only when Discovery or Strategy identifies that specialized knowledge is required. When multiple specialists are needed, they operate in parallel, and their guidance is bundled into the plan together.



Information Flow

Information flows through these teams in a structured cascade:

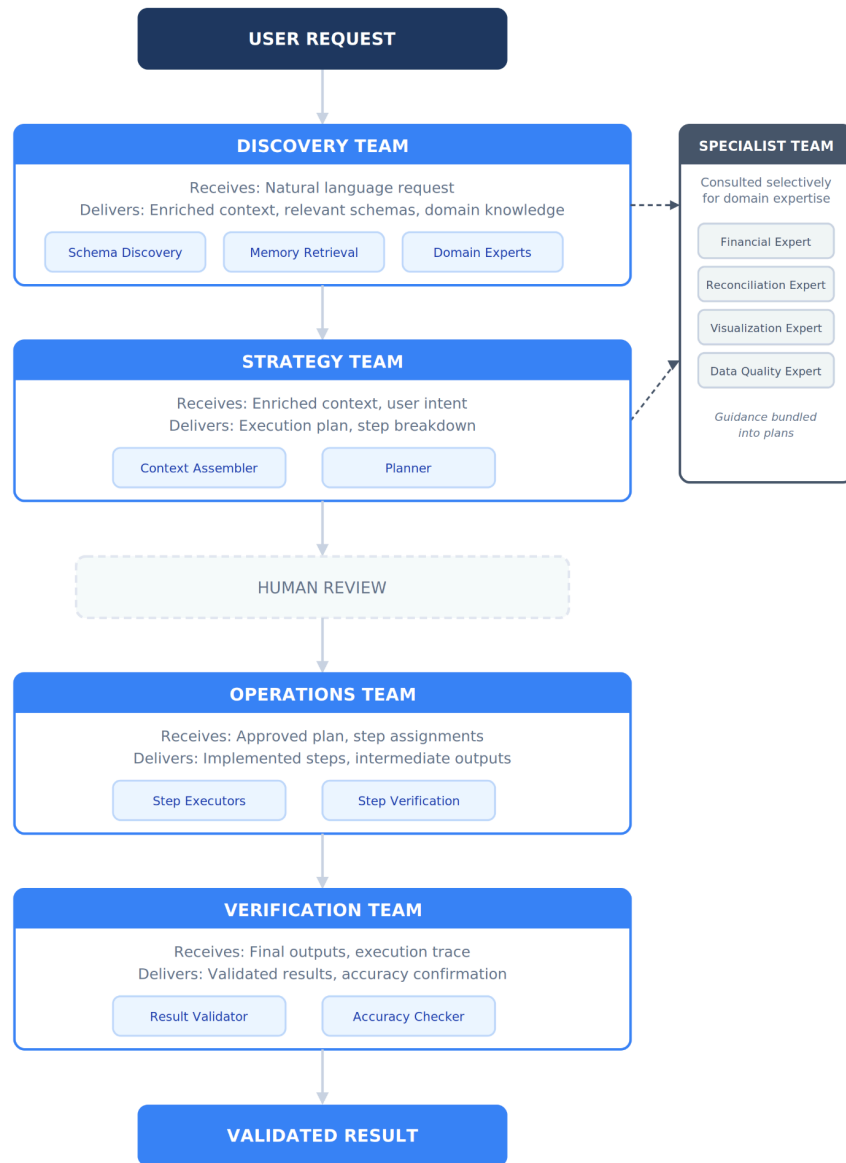


Figure 2: The aidnn orchestration architecture. Requests flow through specialized teams with human review before execution. The Specialist Team provides domain expertise to Discovery and Strategy teams, with guidance bundled into the execution plan.

Each team operates with a defined mandate receiving structured inputs from upstream and delivering structured outputs downstream. The boundaries between teams are explicit, and the handoffs are observable.

This organizational structure is not incidental. It is the mechanism by which aidnn achieves predictability, debuggability, and correctness at scale.



Why Multi-Agent Systems Fail in Practice

The promise of multi-agent AI is compelling: specialized agents collaborate to solve complex tasks. In practice, many systems fail for predictable reasons.

Coordination Overhead Scales Nonlinearly

As the number of agents increases, coordination overhead grows rapidly. Each message is overhead; each handoff is a potential miscommunication. Without structure, the theoretical benefits of specialization drown in unbounded conversation.

Context Contamination and Goal Drift

When every agent sees every message, agents act on irrelevant information, misinterpret intermediate reasoning as facts, and diverge from the intended goal. Goal drift is common: execution agents “improve” the plan midstream, altering assumptions or redefining outputs without explicit approval.

Debugging and Trust Collapse

In enterprise analytics, trust is non-negotiable. If a system cannot explain what it did, why it did it, and which evidence it relied on, it cannot be used for critical decisions. Many multi-agent designs produce results that cannot be reproduced or audited, making them unsuitable for production.

aidnn addresses these issues by using the seven principles below as architectural constraints. They are not slogans - they are design rules that shape how work flows through the system.

The Seven Principles Behind aidnn

1. Structure Over Freedom

Unstructured collaboration feels flexible; in production it becomes chaotic and untraceable.

In early agentic systems, it is natural to let agents “talk it out.” The interaction resembles a brainstorming session: agents propose approaches, debate tradeoffs, revise assumptions, and converge on a solution. This can work in small demos, but it breaks under enterprise constraints: the same question must be answerable tomorrow, by a different user, on a slightly different dataset, with the same expectations of correctness.

Structure turns expertise into repeatable outcomes. Before aidnn produces a plan, it moves through request analysis, metadata discovery, memory retrieval, and domain consultation. Each stage has explicit inputs and outputs, assembling a context



bundle that makes execution predictable. Each stage follows single-responsibility principles, enabling parallel execution where dependencies allow and isolating failures to specific components.

Structured workflows create three benefits. First, they improve predictability: you can expect similar behavior across runs and users. Second, they improve observability: each stage can be logged, monitored, and evaluated independently. Third, they enable improvement: when failures occur, you can pinpoint which stage needs better retrieval, stronger constraints, or different routing.

A structured system can still be flexible. The goal is not to reduce capability, but to channel capability into stable pathways. In practice, structure allows more sophisticated behavior because it reduces the cognitive burden of coordination and makes it safer to add specialized agents over time.

Design implications

- Define explicit workflow phases and treat phase boundaries as contracts.
- Represent intermediate artifacts (context bundle, plan, execution results) as named objects with stable schemas.
- Log per-phase inputs/outputs to enable debugging, evaluation, and regression testing.
- Use user-visible checkpoints (e.g., plan review) to prevent silent drift.
- Use internal functions for deterministic operations, reserving LLM reasoning for genuinely ambiguous steps.

Common anti-patterns

- A single long multi-agent thread where all agents see all messages.
- Agents that decide to “skip ahead” or “change the question” during execution.
- No stable artifacts - only chat transcripts - making results hard to reproduce.
- Relying on prompt length and temperature tweaks instead of architectural constraints.

Adoption checklist

- Can you explain the workflow as a sequence of stages with clear handoffs?
- Do you have stable artifacts for plan and execution (not just text)?
- Can you replay a run and get comparable behavior?
- Can you measure per-stage quality and costs?



2. Hierarchy Tames Complexity

Specialization requires coordination - and coordination costs grow quickly without hierarchy.

As systems scale, the challenge is rarely raw capability; it is coordination. In *The Mythical Man-Month*, Fred Brooks's observation about communication overhead in human teams applies directly to agents: when n participants must coordinate as peers, the number of communication channels grows quadratically. Naive agent swarms waste capacity on negotiation rather than execution.

Hierarchy is not bureaucracy. It is a coordination strategy. aidnn adopts a “surgical team” model where a chief architect (the planner) preserves conceptual integrity and directs specialized agents. Specialists execute within well-defined scopes, and escalation paths are explicit rather than implicit.

Hierarchy also enables safety and cost control. Not every task requires the most powerful model or the richest context. aidnn uses capability tiers across heterogeneous models from multiple providers, selecting each for its strengths: high-capability reasoning models for planning and ambiguity resolution, fast models for routine execution, and escalation to more powerful models when difficulty rises. This structure reduces cost while preserving reliability and avoids single-vendor lock-in.

Finally, hierarchy supports context hygiene through controlled visibility. If every agent sees every message, execution agents may second-guess resolved decisions or hallucinate. Message filtering ensures that each agent receives precisely what it needs - no more and no less - reducing confusion and improving determinism.

Design implications

- Use a central planner (or small planning team) to own global coherence.
- Assign specialized agents narrowly scoped responsibilities and stable interfaces.
- Implement controlled message visibility to prevent context contamination.
- Adopt capability tiers and explicit escalation to balance cost and correctness.

Common anti-patterns

- A mesh of peer agents negotiating continuously without a coordinating authority.
- Execution agents rewriting goals based on partial information.
- One agent with many responsibilities and tools, increasing ambiguity.
- No escalation policy - failures become endless loops or silent degradation.

Adoption checklist

- Is there a single component accountable for conceptual integrity?



- Can you describe what each agent does in one sentence?
- Do agents have strict context boundaries?
- Do you have a defined escalation path for difficult cases?

3. Plans Are Contracts

A plan is a specification. Execution fulfills it without renegotiation.

In enterprise settings, the difference between a helpful prototype and a trustworthy system is the plan. A plan is not a rough outline - it is a contract. It states what will be done, in what order, with which assumptions, and what outputs will be produced.

aidnn treats planning as the chief architect's job. The planner receives the accumulated context bundle and produces a formal specification. Importantly, the plan surfaces assumptions explicitly: date ranges, definitions, filters, and methodological choices. This makes hidden decisions visible before they affect results.

Execution agents then implement the plan exactly. They do not "improve" the specification midstream. This rigidity exists to protect conceptual integrity. Local optimizations made by an execution agent - without full context - often compound into system-level incoherence.

When execution discovers that an assumption is wrong or a dependency is missing, aidnn escalates to replanning instead of patching with local fixes. This preserves traceability and makes behavior more predictable and reviewable.

Design implications

- Pin model versions to ensure reproducibility across runs.
- Enforce strict structured outputs at every step to eliminate parsing ambiguity.
- Represent the plan as a structured artifact with step-level contracts (inputs, outputs, constraints).
- Expose assumptions and uncertainties explicitly in the plan for review.
- Prevent execution agents from modifying plan scope; require explicit replanning.
- Use citations and provenance so each plan element can be traced to evidence.

Common anti-patterns

- Plans that are vague, informal, or embedded only in chat text.
- Execution agents that silently change definitions (e.g., "active customer").
- Accidental scope creep where later steps add new outputs or analyses.
- No provenance - users cannot tell where assumptions came from.



Adoption checklist

- Does every step have a clear input/output contract?
- Are assumptions and definitions explicitly stated and reviewable?
- Can execution be replayed deterministically from the plan?
- Is replanning the only way to change strategy?

4. Leverage Through Information

Small informational inputs can transform downstream work from exploration to implementation.

In *High Output Management*, Andy Grove described managerial leverage as achieving outsized impact through targeted information gathering, decision-making, and nudging. In agentic systems, leverage often comes from the same source: the right information at the right time.

A common failure mode in analytics agents is wasted exploration. The agent spends tokens and time searching for tables, guessing join keys, and iterating on definitions. aidnn reduces exploration by investing in structured context gathering before planning and execution.

This principle appears in the progressive metadata discovery approach: begin with broad semantic retrieval over schema summaries, rerank to increase precision, then retrieve deep technical details only for the most relevant candidates. By narrowing scope step by step, aidnn avoids drowning the model in irrelevant context while still achieving high recall.

Leverage also comes from memory. Organizational knowledge - like “subscription amounts are stored as monthly base amounts” or “use analytics.revenue_summary rather than staging tables” - converts hours of analyst experience into reusable system guidance. aidnn’s memory hierarchy separates ephemeral session memory from persistent organization memory and user preferences, enabling learning without compromising privacy.

Design implications

- Use staged retrieval pipelines (high recall → high precision → deep details).
- Capture and promote reusable insights into organization memory with governance.
- Separate organization memory from user memory; avoid leaking personal preferences.
- Make uncertainty visible and use user feedback to refine future retrieval.



Common anti-patterns

- Dumping entire catalogs or all table schemas into context (“context flood”).
- Keyword-only search that misses semantic relationships.
- No mechanism to learn from corrections or repeated analyst decisions.
- Treating memory as a single undifferentiated blob with no lifecycle or scope.

Adoption checklist

- Do you have a staged retrieval strategy that controls scope and cost?
- Can the system learn durable mappings/definitions from user feedback?
- Is memory scoped (session vs organization vs user)?
- Can users correct ambiguity and see it reflected later?

5. Iterate Tactics, Not Strategy

Critique loops refine execution without corrupting the intent of the plan.

In real analysis, first attempts often fail. Data may be messy, schemas may be inconsistent, and edge cases appear unexpectedly. A production system must therefore iterate - but iteration must be controlled.

aidnn separates strategy from tactics. Strategy is the plan: what will be accomplished and why. Tactics are how a step is executed: the exact code, transforms, or parameter values used to implement a step. Critique loops operate within the tactical layer.

This separation prevents goal drift. An execution agent may attempt several tactical approaches to satisfy a step contract - guided by critique feedback - but it may not redefine what the step means. When tactical iteration reveals that the plan’s assumptions are wrong (e.g., missing column, incorrect definition, join key mismatch, or data unavailable for the requested date range), the system escalates to replanning explicitly.

Runtime consultants support tactical decisions that cannot be resolved at planning time, such as discovering join keys by inspecting data samples. This preserves the plan’s intent while acknowledging that some implementation details require runtime evidence.

For high-stakes outputs where factual accuracy is critical, aidnn employs an LLM council: multiple models independently evaluate claims, and a reasoning model adjudicates based on evidence. This ensemble approach reduces hallucination risk and increases confidence in grounded outputs.



Design implications

- Implement critique loops at the step level with clear success criteria.
- Allow tactical iteration but gate strategic changes behind replanning.
- Introduce runtime consultants for data-dependent questions (e.g., join key discovery).
- Define escalation policies: retry, upgrade model tier, ask user, or replan.

Common anti-patterns

- Iteration that changes the meaning of outputs without user awareness.
- Unlimited retries without a success criterion or escalation policy.
- Treating every failure as a prompt tweak rather than isolating the failing step.
- No runtime evidence collection - agents guess keys and relationships.

Adoption checklist

- Do you have explicit step success criteria for critique?
- Is there a clear boundary between tactical retries and strategic changes?
- Can the system collect runtime evidence safely and deterministically?
- Do repeated failures trigger escalation rather than endless loops?

6. Emergence Over Programming

The most impressive capabilities emerge from integration - not from hard-coded logic.

The strongest agentic systems are not those with the most complex hand-written logic; they are those that combine simple components into a coherent whole. aidnn's capabilities emerge from integrating memory, domain expertise, critique, and execution under a structured plan.

Consider reconciliation tasks. A generic code-writing agent can write joins and transformations, but it lacks organization-specific conventions and domain methodology. When the agent receives memory about column mappings, a domain expert's reconciliation heuristics, and critique that enforces robustness, it produces expert-level work - without being explicitly programmed for that organization.

This compositional approach improves adaptability. As new domains, data sources, or workflows are introduced, the system can extend by adding new experts, new retrieval sources, or new validators - while preserving the overall orchestration structure.

Emergence also encourages modular evaluation. Instead of treating the system as a monolith, you can test and improve components: retrieval quality, planning quality,



step execution reliability, and validation. Improvements compound across the full workflow.

Design implications

- Build simple, testable components that compose (retrieval, planning, execution, critique, SMEs).
- Design interfaces so components can be swapped or improved independently.
- Evaluate at both component level and end-to-end to capture emergent effects.
- Treat organizational memory and SME guidance as first-class inputs to execution.

Common anti-patterns

- Hard-coding complex business logic into prompts and hoping it generalizes.
- A monolithic agent that mixes planning, execution, and validation in one step.
- No way to evaluate or improve components independently.
- Adding more agents without improving orchestration leads to more chaos, not more capability.

Adoption checklist

- Can you add an expert or validator without rewriting the whole workflow?
- Are components independently testable and observable?
- Do you have mechanisms to capture and reuse organization-specific knowledge?
- Do improvements in one layer translate to measurable end-to-end gains?

7. Humans Own the What

Human review is not bureaucracy; it is the foundation of trust and accountability.

In enterprise analytics, the hardest part is rarely producing an answer - it is producing an answer that stakeholders will trust and act on. Trust requires transparency, control, and accountability.

aidnn keeps humans in control of the “what”: definitions, assumptions, and acceptance criteria. The system owns the “how”: retrieving context, producing a plan, executing steps, and validating results. This division of responsibility preserves agency and reduces risk.

Human review is most powerful when it occurs early, before expensive execution. aidnn pauses at the plan stage for approval. Users can correct table selection, definitions (e.g., “active customer”), or scope (e.g., date range). These corrections can then be promoted into organizational memory, improving future runs.



aidnn also supports private interrogation through an Ask-style interface, where users can ask “how did you compute this?” or “why this approach?” without disrupting shared workflows. This encourages understanding while preserving psychological safety and collaboration norms.

Design implications

- Make plans reviewable and modifiable before execution.
- Surface uncertainties and assumptions explicitly for user feedback.
- Provide provenance: citations from schema, memory, and expert guidance.
- Support private explanation channels (Ask/trace) to build trust without friction.

Common anti-patterns

- Fully autonomous execution with no review checkpoints.
- Hidden assumptions embedded in code or prompts.
- No provenance - users cannot verify inputs, joins, or definitions.
- Treating human involvement as an exception rather than a core trust mechanism.

Adoption checklist

- Do users see and approve assumptions before execution?
- Can users trace outputs back to evidence and intermediate artifacts?
- Do corrections feed back into organizational learning?
- Is there a safe way to ask ‘why’ and ‘how’ after results are produced?

Putting the Principles to Work in Enterprise Analytics

Together, these principles form a practical framework for production-grade agentic analytics. They align with enterprise needs in four domains: reliability, governance, cost control, and adoption.

Reliability and Reproducibility

Structured workflows and planning contracts enable repeatable behavior. When results change, teams can attribute the change to data updates, schema changes, memory updates, or model upgrades - rather than opaque agent drift.

Governance and Trust

Human review, provenance, and controlled execution boundaries support governance requirements. Stakeholders can audit what data was used, what assumptions were applied, and how conclusions were derived.



Cost and Performance

Capability tiers, staged retrieval, and pure orchestration patterns reduce token waste and improve latency. The system spends expensive reasoning only where it matters: planning, ambiguity resolution, and critical validation.

Organizational Learning

Memory promotion turns one-off analyses into shared assets. Organizations accumulate definitions, mappings, and conventions over time, creating compounding returns on adoption.

Key Takeaways: What to Evaluate in an Agentic Systems

As enterprises evaluate agentic systems, it is critical to look beyond model benchmarks or demo-level fluency. Production-grade agentic systems succeed or fail based on their architecture, governance model, and ability to scale trust over time.

Key evaluation dimensions include:

- **Workflow structure:** Does the system use explicit stages (context gathering, planning, execution, validation) rather than free-form agent chat?
- **Planning discipline:** Are assumptions, definitions, and scope made explicit and reviewable before execution?
- **Coordination model:** How does the system limit communication overhead and prevent context contamination across agents?
- **Determinism and replayability:** Can analyses be reproduced and debugged when results are questioned?
- **Provenance and explainability:** Can every output be traced back to data sources, definitions, and decisions?
- **Human-in-the-loop controls:** Where do humans review, approve, and override system behavior?
- **Learning and memory:** How does the system capture institutional knowledge without leaking personal context?
- **Failure handling and escalation:** What happens when execution fails repeatedly or assumptions are wrong?
- **Cost and performance management:** How does the platform balance capability, latency, and cost across different tasks?
- **Governance readiness:** Can the system support audit, compliance, and cross-team usage at scale?



Platforms that treat agentic AI as an orchestration discipline - rather than a prompt engineering exercise - are far more likely to deliver durable value in enterprise environments.

Conclusion

aidnn, from Isotopes AI, demonstrates that building reliable agentic AI is fundamentally an engineering discipline. The challenges - coordination overhead, architectural integrity, observability, and trust - are the same challenges organizations have faced for decades in managing complex work.

By applying seven principles - Structure Over Freedom, Hierarchy Tames Complexity, Plans Are Contracts, Leverage Through Information, Iterate Tactics, Not Strategy, Emergence Over Programming, and Humans Own the What - aidnn turns model capability into enterprise-ready outcomes: predictable execution, transparent reasoning, and compounding organizational knowledge.

The result is not a smarter chatbot. It is a structured system that works like a high-performing analyst team: planning deliberately, executing carefully, learning continuously, and always showing its work.