SRAL: A Framework for Evaluating Agentic AI Architectures

Aakash Sharan

Independent Researcher

December 2025

Abstract

The rapid proliferation of LLM-based agents has produced systems with impressive capabilities but inconsistent reliability. Current discourse focuses on what agents can do—tool use, code generation, web browsing—while neglecting the architectural foundations that determine whether they can do these things reliably. This paper introduces SRAL (State-Reason-Act-Learn), a minimal evaluation framework for reasoning about agent architecture. Unlike perception-oriented loops such as Sense-Reason-Act-Learn or Perceive-Reason-Act, SRAL foregrounds *State*—the constructed, persistent world-model that agents must explicitly maintain—as the foundational component upon which reasoning, action, and learning depend. We argue that most agent failures trace not to reasoning or action, but to unmanaged state: context window overflow, lost constraints, and forgotten decisions. SRAL provides both a dependency model (State → Reason → Act → Learn) and an evaluation methodology (four architectural questions applied in sequence). We differentiate SRAL from related frameworks including OODA, ReAct, and enterprise agent architectures, and demonstrate its application as an analytical tool for agent system design.

Keywords: agentic AI, agent architecture, LLM agents, state management, evaluation framework

1 Introduction

The term "agentic AI" has rapidly entered mainstream discourse. Technology vendors announce agent capabilities weekly. Research papers propose increasingly sophisticated agent architectures. Organizations experiment with deploying agents for customer service, code generation, data analysis, and workflow automation.

Yet beneath this activity lies a fundamental confusion. Most conversations about AI agents focus on capabilities. Can the agent browse the web? Can it write and execute code? Can it use external tools? These questions matter, but they address only what an agent can do, not whether it can do it reliably.

Capability and reliability are not the same. An agent that performs impressively in a demonstration may fail unpredictably in production. The difference is rarely the underlying model—the same model powers both the demo and the deployment. The difference is architecture: the structures that manage state, ground reasoning, handle feedback, and enable improvement over time.

This paper argues that the field lacks a shared vocabulary for discussing agent architecture. We evaluate agents by their capabilities—their tool repertoire, their context window size, their benchmark scores—while neglecting the architectural decisions that determine whether those capabilities translate into reliable behavior.

Consider a common failure pattern. An agent is given a complex task requiring multiple steps. It begins well, gathering information and forming a plan. Midway through execution, it makes a decision that contradicts an earlier constraint. When questioned, it has no memory of the constraint—it was lost when the context window filled and earlier content was truncated. The agent's reasoning, given what it remembered, was sound. The failure was not in reasoning but in state management.

This pattern—visible failure in reasoning or action, root cause in state—recurs across agent systems. Yet we lack frameworks for diagnosing it systematically.

1.1 Contributions

This paper makes three contributions:

1. **We introduce SRAL (State-Reason-Act-Learn)**, a framework that foregrounds state management as the foundational component of agent architecture. Unlike perception-oriented frameworks that begin with sensing or observation, SRAL begins with the constructed world-model that agents must explicitly maintain.
2. **We establish a dependency model** where each component relies on its predecessors: Reason depends on State, Act depends on Reason, Learn depends on all three. This ordering enables systematic failure analysis by tracing backward from visible symptoms to root causes.
3. **We provide an evaluation methodology**—four architectural questions applied in sequence—that practitioners can use to analyze any agent system and identify structural weaknesses.

The remainder of this paper is organized as follows. Section 2 reviews related frameworks and positions SRAL against prior work. Section 3 presents the SRAL framework in detail. Section 4 demonstrates SRAL as an evaluation methodology. Section 5 discusses limitations and scope. Section 6 concludes.

2 Background and Related Work

Agent architectures have been studied across multiple disciplines, from military strategy to robotics to artificial intelligence. This section positions SRAL against four influential frameworks: the OODA loop from military decision-making, Sense-Reason-Act-Learn from physical AI, ReAct from LLM research, and Perceive-Reason-Act from enterprise agent design.

2.1 The OODA Loop

The OODA loop—Observe, Orient, Decide, Act—was developed by military strategist John Boyd in the 1970s to describe decision-making in adversarial, time-pressured environments [1]. Boyd argued that competitive advantage accrues to those who can cycle through OODA faster than their opponents, updating their understanding and adapting their actions more rapidly.

OODA has been influential across many fields. However, OODA was developed for human decision-makers who possess inherent capabilities that software agents lack. Humans have persistent memory—a fighter pilot does not forget the threat assessment from thirty seconds ago. Humans learn automatically from experience. OODA assumes these capabilities; it does not address how to provide them.

SRAL addresses systems where memory and learning must be architecturally provided. The "Orient" phase in OODA most closely corresponds to SRAL's State component. But SRAL

makes explicit what OODA leaves implicit: that state must be constructed, managed, and persisted through deliberate architectural choices.

2.2 Sense-Reason-Act-Learn

The formulation "Sense-Reason-Act-Learn" appears in recent industry analyses of physical AI—systems that combine AI reasoning with robotic sensing and actuation. Zinnov's 2025 reports describe machines that can "Sense, Reason, Act, and Learn (SRAL) inside physical environments" [4].

This usage is oriented toward embodied systems: robots, autonomous vehicles, industrial automation. "Sense" refers to perception through cameras, LiDAR, and tactile sensors.

The present framework shares the acronym but differs in its first component and target domain. We use "State" rather than "Sense" to emphasize a distinction critical for software agents:

- **Sense** implies continuous perception—a stream of environmental input that the agent receives.
- **State** implies constructed knowledge—an explicit world-model that the agent builds, maintains, and manages.

For physical AI systems, sensing is fundamental. But LLM-based software agents face a different challenge. They do not lack perception; they receive input through user messages, API responses, and tool outputs. What they lack is persistent state. Without explicit architecture for state management, context is lost as conversations lengthen, and the agent cannot maintain coherent understanding across extended tasks.

2.3 ReAct

ReAct, introduced by Yao et al. [3], proposed interleaving reasoning and acting in language models. The ReAct pattern alternates between "Thought" (the model verbalizing its reasoning), "Action" (calling a tool), and "Observation" (the result fed back to the model).

ReAct addressed a critical problem: language models reasoning in isolation tend to hallucinate. By interleaving action and observation, ReAct grounds reasoning in environmental feedback.

SRAL builds on this contribution. The Act component explicitly incorporates the insight that action should inform reasoning, not merely execute it. However, ReAct does not address state persistence—state exists only in the context window. Similarly, ReAct operates within a single session, providing no mechanism for learning across sessions.

SRAL extends ReAct's contribution by adding explicit state management (beyond the context window) and learning (beyond a single session).

2.4 Perceive-Reason-Act

AWS Prescriptive Guidance describes agent architecture as a "perceive, reason, act" cycle [2]. The Perceive module transforms raw input into structured representations. The Reason module—encompassing memory, knowledge bases, goals, and decision-making—evaluates context and determines actions. The Act module executes actions and captures feedback.

This architecture provides useful implementation guidance. However, it bundles state management into the Reason module rather than treating it as a separate foundational concern.

The bundling matters because state failures are often invisible to reasoning. When state is incomplete, the reasoning module does not receive an error message. It simply reasons from

whatever state it has access to, unaware that information is missing.

SRAL separates State from Reason precisely to make state failures visible as an architectural concern.

2.5 Summary of Positioning

Framework	Origin	First Component	SRAL's Distinction
OODA	Military strategy	Observe	Explicit state architecture
Sense-RAL	Physical AI	Sense	State as constructed world-model
ReAct	LLM research	Thought	State persistence; learning
Perceive-Reason-Act	Enterprise agents	Perceive	State as separate layer

Table 1: SRAL positioned against related frameworks

3 The SRAL Framework

This section presents SRAL, a framework for evaluating the architectural foundations of agentic AI systems. SRAL consists of four components—State, Reason, Act, and Learn—arranged in a strict dependency order.

3.1 Overview and Dependency Model

SRAL proposes that agent architectures be evaluated along four dimensions, in a specific order:

$$\textbf{State} \rightarrow \textbf{Reason} \rightarrow \textbf{Act} \rightarrow \textbf{Learn}$$

This ordering reflects causal dependencies:

- **Reason depends on State.** An agent cannot reason coherently about a situation it has forgotten.
- **Act depends on Reason.** Tool use without strategic reasoning produces thrashing, not progress.
- **Learn depends on all three.** Improvement requires observing consequences of actions, which requires acting, which requires reasoning, which requires state.

This dependency structure has practical implications for failure analysis. When an agent fails, the visible symptom typically appears in Act or Reason. However, tracing backward often reveals that the root cause lies in State: a lost constraint, a truncated context, a forgotten decision.

3.2 State: The Constructed World-Model

Definition. State refers to the constructed, evolving world-model that an agent maintains throughout task execution. This includes the agent's understanding of the current task, its memory of previous steps, constraints that must be satisfied, and context carried forward from earlier interactions.

State is distinct from perception. Perception is the receipt of raw input; state is what the agent *knows* after processing, structuring, and storing that input.

The Architectural Question. *Is state explicit and managed, or implicit and fragile?*

Most contemporary LLM agents rely on the context window as their sole form of memory. Context windows have finite length; when they fill, information is truncated. The agent then reasons from incomplete state, typically without awareness that information has been lost.

Failure Modes:

- *Context overflow.* Long-horizon tasks exceed the context window, causing early information to be lost.
- *Implicit state assumptions.* Different system components hold inconsistent assumptions.
- *Institutional memory failure.* Agents lack persistent state across sessions and cannot accumulate knowledge.

Design Implications. Robust state architecture requires explicit decisions: What information must persist? Where is it stored? How will it survive context limits?

The institutional analogy is instructive. A team that does not document its decisions eventually forgets why it made them. State in agent systems plays the same role that documentation and institutional memory play in human organizations.

3.3 Reason: Grounded Deliberation

Definition. Reasoning is the process by which an agent moves from understanding to intention. This encompasses goal decomposition, planning, replanning, exception handling, and selection among alternatives.

The Architectural Question. *Is reasoning grounded in reality, or floating in abstraction?*

The quality of reasoning is bounded by the quality of state. An agent reasoning from incomplete state will produce plans that are internally coherent but externally invalid.

Failure Modes:

- *Hallucination cascade.* The model invents a fact, reasons from it, and arrives at factually incorrect conclusions.
- *Reasoning drift.* Accumulated small errors compound over extended interactions.
- *Abstraction without verification.* The agent constructs plans without checking whether preconditions hold.

Design Implications. Grounded reasoning requires mechanisms connecting deliberation to environmental feedback, such as the ReAct pattern of interleaving reasoning with action and observation.

3.4 Act: The Environmental Interface

Definition. Action encompasses all mechanisms by which the agent affects its environment: tool use, API calls, file operations, and communication.

Action is where the agent's internal representations are tested against the world.

The Architectural Question. *Does action inform reasoning, or merely execute it?*

This distinguishes open-loop from closed-loop architectures. In closed-loop execution, the agent observes results and updates reasoning before proceeding.

Failure Modes:

- *Tool misuse.* Wrong parameters, wrong sequencing, failure to handle errors.
- *Observation neglect.* The agent proceeds with predetermined plans regardless of results.
- *Feedback loop absence.* Action results do not update state or inform reasoning.

Design Implications. Effective action architecture creates a closed loop: action produces observation, observation updates state, updated state improves reasoning. Tools do not make agents intelligent; they expose how intelligent the agent already is.

3.5 Learn: Architectural Improvement

Definition. Learning refers to mechanisms by which an agent incorporates feedback, adjusts strategies, and improves over time.

Learning is the weakest component in most contemporary agent systems. Most agents do not learn; each conversation begins fresh.

The Architectural Question. *Is learning architectural, or accidental?*

Failure Modes:

- *Repeated mistakes.* Without memory of past failures, agents make the same errors repeatedly.
- *No transfer.* Success on one task does not inform performance on similar tasks.
- *Perpetual novice state.* The agent never develops expertise.

Design Implications. Learning can occur at multiple timescales: within-session (adjusting during a task), cross-session (remembering across interactions), and cross-task (transferring to related problems). Building architectural support for learning remains among the most important open problems in agent design.

3.6 Summary

Component	Definition	Architectural Question
State	Constructed, persistent world-model	Explicit and managed, or implicit and fragile?
Reason	Grounded deliberation	Grounded in reality, or floating in abstraction?
Act	Environmental interface	Informs reasoning, or merely executes?
Learn	Improvement mechanisms	Architectural, or accidental?

Table 2: The four SRAL components

4 SRAL as Evaluation Methodology

Beyond defining components, SRAL provides a practical methodology for evaluating agent systems.

4.1 The Four Questions

When evaluating any agent system, apply these questions in sequence:

1. **State:** What does it remember? Is state explicit and managed?
2. **Reason:** How does it decide? Is reasoning grounded in reality?
3. **Act:** How does it affect the world? Does action inform reasoning?
4. **Learn:** How does it improve? Is learning architectural?

The sequence matters. State questions come first because state failures undermine everything downstream.

4.2 Failure Tracing: A Worked Example

Consider an agent tasked with producing a report under 2,000 words with at least three academic sources. After forty minutes of research, it produces a 4,500-word report with only two sources—one explicitly flagged to avoid.

Applying SRAL's backward tracing:

Act analysis: The agent produced output. Tools worked. Failure is not in the action layer.

Reason analysis: The reasoning shows coherent planning, but makes no reference to word limits or source requirements. Why?

State analysis: After many tool calls, the context window filled. Automatic truncation removed the earliest messages—including the original requirements.

Root cause: State failure (context overflow). The visible symptoms appeared in output and reasoning, but the architectural root cause was absence of state preservation mechanisms.

Remedy: Explicit state management—a pinned "requirements" object persisting regardless of context length.

4.3 Design Application

SRAL guides system design by ordering concerns:

1. Design state architecture first.
2. Design reasoning patterns second.
3. Design action interfaces third.
4. Design learning mechanisms last.

This ordering prevents building sophisticated reasoning on fragile state foundations.

5 Discussion

5.1 Limitations and Non-Goals

SRAL is an evaluation and design framework. It is not:

- **An algorithm.** SRAL identifies what must be addressed, not how.
- **A guarantee of intelligence.** Satisfying SRAL criteria does not guarantee effectiveness.
- **A replacement for model capability.** Architecture matters in addition to model capability, not instead of it.
- **A complete theory of agency.** SRAL is minimal by design and does not address multi-agent coordination, safety, or efficiency.

5.2 Scope

SRAL is designed for LLM-based software agents operating through tool use and natural language communication. For physical AI systems where continuous sensing is fundamental, the Sense-Reason-Act-Learn formulation may be more appropriate.

5.3 Future Work

- **Empirical validation:** Applying SRAL to existing frameworks (LangChain, AutoGen, CrewAI).
- **Benchmark development:** SRAL-informed benchmarks for state management, reasoning grounding, and learning.
- **Multi-agent extension:** Addressing shared state and coordinated reasoning.

6 Conclusion

The proliferation of agentic AI has outpaced our vocabulary for evaluating it. We measure agents by capabilities while neglecting architectural foundations.

This paper introduced SRAL (State-Reason-Act-Learn), a minimal framework for evaluating agent architecture. SRAL makes three contributions:

First, it foregrounds state as the foundational component. State is not what the agent receives; it is what the agent knows—processed, structured, and persisted.

Second, it establishes a dependency model enabling failure analysis. When an agent fails, tracing backward through the chain typically reveals root causes in state that manifest as symptoms in reasoning or action.

Third, it provides an evaluation methodology: four questions, applied in sequence, directing attention to foundations before features.

The hype cycle rewards capability. But capability without architecture is fragile. The agent that impresses in a demo may fail in production—not because the model is inadequate, but because the architecture cannot maintain state, ground reasoning, incorporate feedback, or learn from experience.

Architecture endures. The teams that understand this will build the agents that actually work.

The model is not the agent. The architecture is.

References

[1] Boyd, J. R. (1976). Destruction and Creation. Unpublished manuscript.

[2] AWS (2024). Foundations of Agentic AI on AWS. AWS Prescriptive Guidance. https://docs.aws.amazon.com/prescriptive-guidance/latest/agentic-ai-foundations/

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

[4] Zinnov (2025). Physical AI: The Next USD 300 Bn Opportunity for Tech Services. Zinnov Report.