

Closing the Loop

Engineering agentic loops that verify their own progress, stop when they should, and keep working when the network does not.

Most AI agents fail the same way. Not because the model is weak, but because the loop around it was never engineered. This paper describes the anatomy of a production agentic loop, the failure modes that break it, the design rules that hold it together, and five patterns you can deploy this quarter.

Joseph C. McGinty Jr.

Founder and Research Lead, ResilientMind AI
Service-Disabled Veteran-Owned Small Business

01 Executive summary

An agentic loop is a system in which a model repeatedly observes its environment, decides on an action, takes it through a tool, checks the result, and decides whether to continue. That repetition is what separates an agent from a chatbot. It is also what makes agents fragile: every pass through the loop is another chance to drift, compound an error, or spend money with nothing to show for it.

The industry conversation centers on models. The operational problems sit in the loop. Three findings frame this paper:

- **Reliability compounds against you.** A step that succeeds 95% of the time succeeds only about 36% of the time across twenty consecutive steps. Long loops require verification at every step, not a smarter model at the end.
- **The model should not grade its own work.** Loops that stop when the model says "done" are loops that stop when the model is confident, which is not the same thing. Termination belongs to external checks: tests, schemas, ground truth, or a human.
- **Autonomy must be bounded and survivable.** Production loops need hard budgets, gates on irreversible actions, an audit trail, and a defined behavior when connectivity, tools, or upstream services disappear.

Bottom line. The difference between an impressive demo and a dependable system is not the model. It is whether the loop knows how to check, when to stop, and what to do when the world fails underneath it.

02 What an agentic loop is

A pipeline runs a fixed sequence once. A loop decides its next step based on what just happened. Every agentic loop, whatever the framework, reduces to five stages:

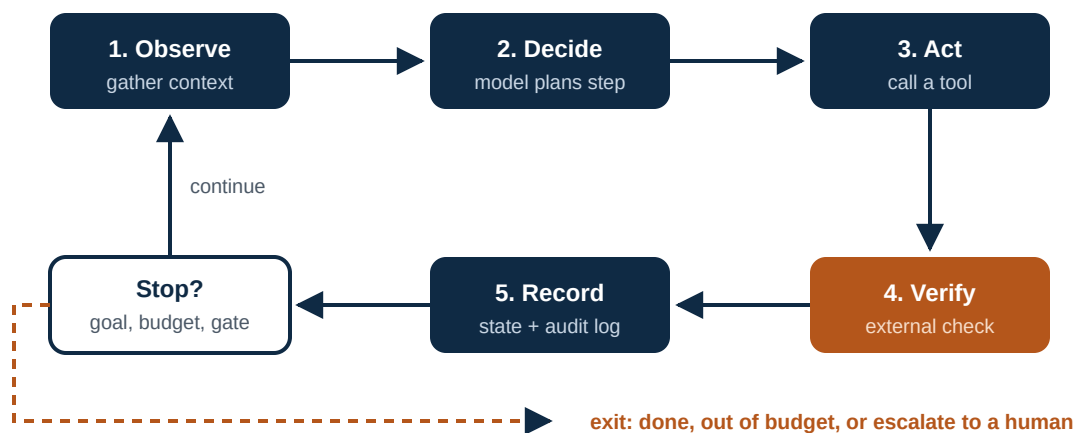


Figure 1. The five-stage agentic loop. Verification and the stop decision sit outside the model.

Around those stages sit the components that determine whether the loop is safe to leave running:

Component	Role
Goal and success criteria	A definition of "done" that a machine can check. "Improve the code" is not a goal. "All tests pass and coverage does not drop" is.
Context assembly	What the model sees on each pass: the goal, recent results, relevant state. Too little and it repeats itself; too much and it loses the thread.
Tools	The only way the loop touches the world. Each tool needs defined permissions, inputs, and failure behavior.
Verifier	An independent check on each result: a test suite, a schema, a comparison against source data, a second model with a different role, or a person.
State and memory	Durable record of what has been tried, what worked, and where the loop is, so it can resume after a crash instead of starting over.
Governor	Hard limits on steps, time, and spend; gates on irreversible actions; the authority to stop the loop regardless of what the model wants.

03 How loops fail

Agent failures are rarely dramatic. They are quiet, cumulative, and expensive. Eight failure modes account for most of what breaks in practice.

Failure mode	What happens	Engineering control
Compounding error	Small per-step error rates multiply across long runs until the outcome is unreliable.	Verify every step; keep loops short; checkpoint known-good states.
Self-grading	The model decides it is finished because it is confident, not because the work is correct.	Termination owned by external checks, never by the model alone.
Runaway iteration	The loop retries the same failing approach indefinitely.	Step, time, and cost budgets; detect repeated actions and force a change or exit.
Goal drift	Over many passes the loop optimizes for a proxy and loses the original intent.	Re-inject the original goal every pass; score against it, not against the last step.
Context rot	Accumulated history crowds out what matters; the model forgets constraints.	Summarize and prune state; keep constraints pinned in every prompt.
Unsafe side effects	A tool call deletes, sends, pays, or deploys something that cannot be undone.	Idempotent tools; dry-run modes; human approval on irreversible actions.
Silent failure	A tool errors or returns empty and the loop proceeds as if it succeeded.	Treat every tool result as untrusted; fail loudly; log everything.
Dependency collapse	The model API, network, or a key service goes down and the loop stops cold, losing work in progress.	Persist state locally; queue actions; degrade to local inference or a safe hold.

The arithmetic of long loops. If each step is 95% reliable, a 20-step run succeeds about 36% of the time (0.95^{20}). At 99% per step, it is about 82%. The lesson is not "buy a better model." It is: verify each step so errors are caught where they happen, and design loops that are short, resumable, and checkable.

04 Seven design rules

1. **Define done in a form a machine can check.** If you cannot write the test, you are not ready to automate the loop.
2. **Separate the maker from the checker.** The component that produces work should not be the one that approves it.
3. **Bound everything.** Maximum steps, maximum wall-clock time, maximum spend. Exiting on a budget is a successful outcome, not an error.
4. **Gate the irreversible.** Reading is free. Writing to production, sending, purchasing, and deleting require a human or a policy check.
5. **Make every action replayable.** Persist state and write an audit trail so any run can be resumed, inspected, or reconstructed after the fact.

6. **Fail loudly and early.** An error that stops the loop is cheap. An error that the loop papers over is the expensive one.
7. **Plan for disconnection.** Decide in advance what the loop does when the cloud, the network, or a dependency disappears: pause safely, queue work, or continue on local models.

Resilience at the edge

Most agent frameworks assume a stable connection to a hosted model. Field operations, emergency management, industrial sites, and defense environments cannot make that assumption. In denied, degraded, intermittent, or limited-bandwidth (DDIL) conditions, an agentic loop must keep its state on the device, make bounded decisions with local inference, record what it did, and reconcile when connectivity returns. The question for these environments is not how capable the agent is when everything works. It is what the agent does in the minutes after something breaks. That is decision continuity, and it has to be designed in from the first line, not bolted on later.

05 Five practical patterns

Each pattern below lists the goal, the loop, the external check, and the stop condition. All five can be built with current tools.

SOFTWARE

1. Test-gated code improvement

Goal	Fix a failing build or implement a small, specified change.
Loop	Read failing tests, propose a patch, apply it in an isolated branch, run the suite, read new failures.
Check	The test suite, linter, and type checker. Not the model's opinion.
Stop	All checks pass, or N iterations reached. A person reviews the diff before merge.

CONTENT

2. Source-verified publishing

Goal	Produce a daily brief or article from defined sources.
Loop	Collect sources, draft, extract every factual claim, check each claim against the source text, revise or cut what does not match.
Check	Claim-to-source matching. Unsupported claims are removed, not softened.
Stop	Every claim traced to a source, then publish. A failed check holds the post instead of publishing it.

OPERATIONS

3. Read-only incident triage

Goal	Shorten time to diagnosis when an alert fires.
Loop	Pull logs and metrics, form a hypothesis, run read-only diagnostics to confirm or reject it, repeat.
Check	Each hypothesis must be supported by a specific log line or metric.
Stop	A confirmed root cause and a proposed fix. Remediation waits for human approval.

DATA

4. Self-quarantining data intake

Goal	Ingest records from external partners without corrupting downstream systems.
Loop	Validate each batch, attempt automated repair of known error types, re-validate.
Check	Schema and business-rule validation on every record.
Stop	Clean records load; anything that fails twice goes to quarantine with a report to a person.

EDGE

5. Disconnected field operations

Goal	Keep a site or team operating when connectivity drops.
Loop	Read local sensors and inputs, decide with an on-device model inside a fixed policy, act, log locally.
Check	A policy layer that permits only pre-approved actions while disconnected.
Stop	On reconnection, sync the log, reconcile state, and hand anything outside policy to a person.

06 A maturity model

Level	Name	Description
0	Prompt	A person asks, a model answers. No loop.
1	Chain	A fixed sequence of model calls. Predictable, but cannot adapt.
2	Tool loop	The model chooses tools and iterates. Capable, but stops when it feels finished.
3	Verified loop	External checks decide progress and completion.
4	Governed loop	Budgets, approval gates, audit trail, and resumable state.
5	Resilient loop	Keeps operating safely through outages and disconnection, then reconciles.

Many agents in use today stop at Level 2. The work that matters is moving them to Level 4, and, for any system that supports operations in the field, to Level 5.

07 Readiness checklist

Before letting a loop run unattended, every answer below should be yes.

- ☐ Can a machine check whether the goal has been met?
- ☐ Is the verifier independent of the model producing the work?
- ☐ Are there hard limits on steps, time, and cost?
- ☐ Does every irreversible action require approval or a policy check?

- ☐ Are tool failures detected and surfaced, never ignored?
- ☐ Is the original goal restated to the model on every pass?
- ☐ Can a crashed run resume from its last checkpoint?
- ☐ Is every action written to an audit log a person can read?
- ☐ Is there a defined behavior when the model API or network is unavailable?
- ☐ Does someone own reviewing what the loop did last week?

08 Conclusion

The next phase of AI adoption will not be decided by which organization has access to the best model. Models are converging and widely available. It will be decided by who engineers the loop: who verifies, who bounds, who audits, and who keeps working when the infrastructure underneath fails. Those are systems-engineering problems with systems-engineering answers. Treat the loop as infrastructure, and agents become something you can depend on. Treat it as a prompt, and they remain a demo.

About the author. Joseph C. McGinty Jr. is founder and research lead of ResilientMind AI, a service-disabled veteran-owned small business building sovereign, offline-first AI infrastructure for degraded and disconnected environments. He brings 20+ years spanning government and corporate work.

Contact: joseph@resilientmindai.com | (724) 875-4037 | ariaos.dev | commandroomai.com | linkedin.com/in/joseph-mcginty-jr
© 2026 ResilientMind AI. May be shared in full with attribution.