CLOUDNIX TECHNICAL REPORT · CNX-TR-2026-05

Engineering Supervised Agentic Workflows

Jeya Kirubakaran, Ajmal Abubacker, Avinash KS and Avinash Nishant

Cloudnix Software Labs, Bengaluru, India

CNX-TR-2026-05 · Version 1.1 · November 2026

Abstract

Agents that plan, call tools and write are easy to demonstrate and hard to run: in production they drift, loop, invent facts and links, overrun budgets and leave no auditable record. We describe the patterns behind agentic systems we operate: a stage-based editorial pipeline with bounded review rounds and human approval; cross-provider maker and checker roles; grounding in a shared rulebook and a verified fact list with link validation; map-reduce generation over whole knowledge bases under a strict time budget; schema-constrained extraction confirmed by people and reused; and per-call usage logging. We relate these to failure modes of production agents and to AI risk-management practice, and close with a case study and a checklist.

Keywords: *AI agents; agentic workflows; LLM; human-in-the-loop; maker-checker; grounding; map-reduce; structured output; JSON Schema; cost control; AI governance*

Contents

1. Executive summary
2. Failure modes of production agents
3. A supervised pipeline
4. Cross-provider maker–checker
5. Grounding
6. Map-reduce generation
7. Schema-constrained extraction
8. Budgets, logging and cost
9. Security and safety
10. Case study: an editorial engine
11. Checklist: 15 questions
12. References

1 Executive summary

Agents that plan, call tools and write are easy to demonstrate and hard to run. In production they drift, loop, invent facts and links, overrun budgets and leave no record anyone can audit. This report describes the patterns behind agentic systems we operate: an editorial pipeline in which agents brief, write, review, illustrate and publish under supervision; map-reduce generation pipelines that produce knowledge graphs, component maps, requirements and test cases from whole knowledge bases; and schema-constrained extraction confirmed by people. The common thread is supervision: stages with bounded retries, a different model provider checking each draft, a shared rulebook and verified facts, budgets in time and tokens, and a person at the gate.

Principles.

1. **Stages, not loops.** Each job has a stage, a limit and a next step.
2. **The maker never checks itself.** A different provider reviews.
3. **Ground every claim.** A rulebook and a verified fact list bound what may be said.
4. **Structure every output.** Schemas, not free text, cross system boundaries.
5. **Budget everything.** Time per request, rounds per draft, tokens per call.
6. **Record everything.** Every call logged with tokens and images; every decision kept.
7. **People decide.** Nothing reaches the world without approval.

2 Failure modes of production agents

Failure	Symptom	Pattern (chapter)
Drift	Quality degrades over long runs	Stage pipeline with bounded retries (3)
Self-confirmation	A model approves its own mistakes	Cross-provider maker–checker (4)
Hallucinated claims	Invented features, figures and links	Rulebook, verified facts, link checks (5)
Context limits	Knowledge bases larger than any prompt	Map-reduce generation (6)
Unusable output	Free text breaks integrations	Schema-constrained extraction (7)
Runaway cost and latency	Callers time out; bills surprise	Budgets and usage logs (8)

3 A supervised pipeline

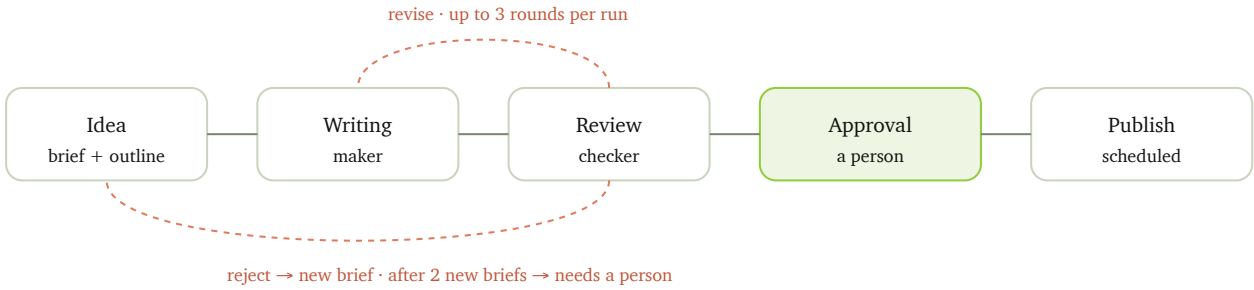


Figure 1. Stages, bounded loops and a person at the gate.

Each stage is a job with its own entry criteria. A run of the scheduler takes the first job that has work: a draft to fix, an idea sent back, or a new idea. The checker may approve, ask for revision or reject. Revision loops are capped at three rounds per run; an article that fails three runs returns to the idea stage for a new brief; an idea that fails after two new briefs is flagged for a person. Every article records its stage, the reason it is there and its history, so a board can show what is happening and why.

3.1 *Scheduling*

One job runs every three hours; publishing of approved work runs every five minutes. Long, unattended loops are not possible by construction.

4 Cross-provider maker–checker

Models share their own blind spots. The provider that writes is never the provider that reviews; the two swap roles each run. Roles are explicit per provider: maker (writes briefs and drafts), checker (scores against the rulebook) and a small model for image plans and photo selection. Over the period measured, 591 model calls were logged, 341 on one provider and 250 on the other [1, 2].

Role	Task	Model class
Maker	briefs, outlines, drafts	large model
Checker	review against rules; approve, revise, reject with notes	large model, other provider
Small	image plans, photo choice	small model

5 Grounding

Every agent reads the same configuration: a rulebook (83 rules, edited by the editorial team), a verified feature list that bounds what may be claimed, a calendar, writer personas and a library of real screenshots. Prompts include only the feature areas, help articles and screenshots nearest the brief; the shared part of the prompt is cached [3]. After generation, every link is checked against real pages and removed if it is not one.

Why a rulebook and not a longer prompt. Rules are edited by the people who own the output, kept in one place, and applied identically by maker and checker, so a correction made once applies to every later run.

6 Map-reduce generation

Generating a knowledge graph, a component map, requirements or test cases requires reading far more than one prompt holds. Marker fields on a request route it to one of five pipelines; each maps over retrieved chunks on local models, caching per chunk, then reduces on the provider the project chose (its own key, or ours).

```
map:    for chunk in retrieve(project, scope): notes[chunk] = cache or local_model(prompt_map,
chunk)
reduce: result = provider.generate(prompt_reduce, notes)           # JSON, repaired if needed
budget: all attempts share 280 s; no new call with < 10 s left; 2 retries on 408/429/5xx
```

Listing 1. Pseudocode.

The budget keeps the whole request inside the caller's own timeout: one attempt may use all remaining time so long answers finish, and when too little remains the request ends with a clear error instead of overrunning.

7 Schema-constrained extraction

When output crosses into another system it must be structured. Import mapping asks a model to map source columns to destinations under a strict JSON Schema [4] with an enumerated set of ten display types and exactly one title; a person confirms or edits the result; the confirmed map is saved and reused by scheduled syncs without calling the model again.

```
{ "mappings": [ { "source_field": string,
                  "destination": "TITLE" | field_id | "NEW" | "SKIP",
                  "display_type": enum(10) } ] }       additionalProperties: false
```

Listing 2. Example exchange or schema.

8 Budgets, logging and cost

- Every call writes a usage record: provider, model, role, tokens in and out, images.
- Costs are attributed per article or per request.
- Time budgets per request, round limits per draft, and caching of shared prompt prefixes keep cost bounded.
- One log per tool per day keeps investigations simple.

9 Security and safety

Risk	Control
Prompt injection via retrieved content	Content treated as data; outputs checked before they act [5]
Unverified claims	Verified fact list; reviewer model; editor approval
Unsafe links	Links checked against real pages
Key exposure	Keys in server-side configuration with restricted permissions; masked in logs
Runaway spend	Budgets, round limits, usage logs

The approach aligns with the governance functions of the NIST AI Risk Management Framework [6]: map, measure, manage and govern.

10 xAgent roles and responsibilities

Treating “the agent” as one entity hides where things go wrong. We separate roles, each with its own prompt, model class, inputs, outputs and failure handling.

Role	Input	Output	On failure
Ideator	sources list, calendar, verified features	brief and outline (JSON)	new brief; flagged after two attempts
Writer (maker)	brief, persona, rulebook, nearest context	structured article blocks	revision with checker notes
Checker	draft, rulebook, verified features	approve / revise / reject with notes	reject returns to ideation
Image planner (small)	approved text	image plan per section	text-only fallback
Photo selector (small)	image plan, licensed photo search	chosen photos with credits	illustration instead of photo
Publisher	approved article	post with category, tags, author, media	retried on the next five-minute run

10.1 *Why small models for small jobs*

Image planning and photo choice are short, frequent decisions. Running them on small models reduces cost without touching the quality of the writing, which stays on large models.

11 xState and data model

Every unit of work is a record with a stage, the reason it is in that stage and an append-only history. This is what makes a pipeline inspectable: the board is simply a view of these records.

```
article = { slug, stage: idea | writing | review | approval | published | stuck,
            stage_note: "why it is here", persona, brief, draft_ref,
            history: [ { at, from, to, by: agent | person, note } ],
            usage:   [ { call_id, provider, model, role, tokens_in, tokens_out, images } ] }
```

Listing 3. Example exchange or schema.

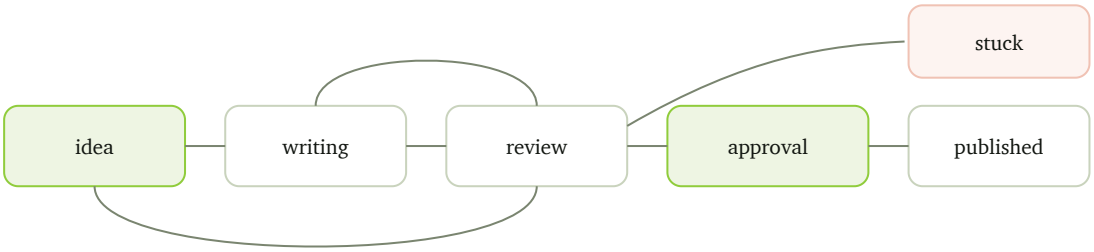


Figure 2. Article states. Every transition is written to the history with who made it and why.

11.1 *Concurrency*

Each scheduler run takes one job and holds a lock on the article list, so two runs never edit the same article. The review page writes decisions to a separate file that the publisher applies, so a person's action and an agent's action never collide.

12 xPrompt design that survives change

- **Shared prefix, variable suffix.** The rulebook, personas and verified features form a stable prefix that is cached; only the brief and nearest context vary.
- **Nearest context only.** Feature areas, help articles and screenshots are ranked by closeness to the brief; only the top few are included.
- **Structured outputs.** Drafts are returned as typed blocks (headings, paragraphs, lists, images), validated before rendering.
- **Model names in configuration.** Each provider and role reads its model from configuration, so upgrading a model is a configuration change, not a code change.
- **Notes as data.** Checker notes and editor notes are stored and fed back on the next attempt, so revisions are targeted.

12.1 *Persona and voice*

Writer personas define tone, habits, topics and a photo description. Personas never claim invented personal facts; the rulebook enforces this.

13 xQuality assurance for agent output

Check	Where	Effect
Rulebook compliance	checker model	revise or reject with notes
Claims against verified features	checker model	unsupported claims removed
Link validity	deterministic check	non-pages removed
Image suitability	small model + rules	off-topic or non-local images replaced
Title style	rulebook	no colon titles, no title case clichés
Human approval	editor on the board	publish, back to writing, back to idea

13.1 *Measuring the pipeline*

Useful measures: share of drafts approved in the first round, average review rounds, ideas re-briefed, articles stuck, tokens per published article and time from idea to publication. Each is computed from the history and usage records without extra instrumentation.

14 xGeneration pipelines over knowledge bases

Pipeline	Marker	Output
Knowledge graph	group type	topics and relations across the project's knowledge base
Components	component type	one level of modules, sub-modules or features
Requirements	generation type	features, each with child requirements, from selected documents
Test cases	generation type	test cases for given requirements, keeping test-data variables
Image to markdown	extraction type	semantic markdown from one image

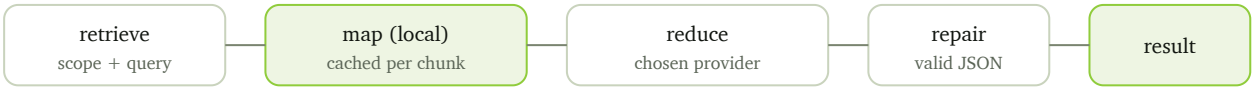


Figure 3. Map-reduce generation with a map cache and JSON repair.

15 Case study: an editorial engine

A content programme for a commerce platform, run by agents with an editor at the gate.

The engine produces illustrated articles in the voices of a small set of writer personas. Each article moves through idea, writing, review, approval and publishing; images mix generated illustrations, licensed photos with credits, diagrams and real screenshots, with the lead image rotating between seven kinds. At the time of writing the board held 23 ideas, 9 articles awaiting approval and 8 published.

15.1 *What it changed*

Content is produced continuously without the editor drafting; the editor's time goes into approvals and rules rather than first drafts; and every claim is bounded by the verified list.

16 Checklist: 15 questions

1	Is every agent job a stage with an exit?
2	Are retries capped?
3	Does a different provider review?
4	Is there one rulebook for maker and checker?
5	Are claims bounded by a verified list?
6	Are links validated?
7	Is structured output enforced by schema?
8	Does a person confirm new mappings?
9	Is confirmed work reused instead of regenerated?
10	Does every request have a time budget?
11	Is every call logged with tokens?
12	Are costs attributed per output?
13	Is anything published without approval?
14	Can a failing idea reach a person automatically?
15	Who owns the rules?

17 xA worked run, end to end

The table follows one article through a real-shaped run. Token counts are illustrative; the sequence, roles and limits are those of the production pipeline.

#	Step	Actor	Result	Recorded
1	Pick job	scheduler	oldest ready idea	lock taken
2	Load context	system	rulebook, persona, nearest feature areas and screenshots	prefix cache hit
3	Draft	maker, provider A	typed blocks, ~1,200 words	tokens in/out
4	Review	checker, provider B	revise: two notes (unsupported claim, title style)	notes saved
5	Redraft	maker, provider A	notes applied	tokens
6	Review	checker, provider B	approve	stage → approval
7	Images	small model	plan: hero infographic, two in-body photos, one diagram	images
8	Links	deterministic	one link removed (not a real page)	change logged
9	Approval	editor	publish	decision file
10	Publish	publisher (5-min job)	post with category, tags, author, featured image	published list

18 xDesigning the rulebook

A rulebook is code for language models: it should be specific, testable and owned. Rules that worked best were concrete and checkable.

Weak rule	Strong rule
Write engagingly	600–900 words; one idea per paragraph; no more than one list per article
Be accurate	Mention only features in the verified list; never quote prices or numbers not in it
Good titles	No colons; sentence case; no “The ultimate guide”
Use relevant images	Hero must match the topic; photos must show the target region; no text in generated images
Sound human	Write in the persona’s voice; never invent personal facts for a persona

Every editor note that recurs becomes a rule; a previous version of the rulebook is kept so changes can be reviewed.

19 xFailure handling and recovery

Failure	Detection	Response
Provider timeout or 5xx	HTTP status	retry within budget; otherwise leave job for next run
Rate limit (429)	HTTP status	back off; retry within budget
Invalid structured output	schema validation	JSON repair; else retry once
Checker rejects	decision	new brief; flag after two
Image fetch fails	status / licence check	fall back to illustration
Publish fails	CMS response	retried on next five-minute run
Run crashes mid-job	lock age	lock released; job resumes from last saved stage

20 xCost model

Cost per output = Σ over calls (tokens in × input price + tokens out × output price) + images × image price. Three levers matter most: caching the shared prefix (largest input saving), keeping review rounds low (each round adds a maker and a checker call) and using small models for small decisions.

```
cost(article) = Σ_calls ( in_tokens·p_in(model) + out_tokens·p_out(model) ) + images·p_img
rounds_effect ≈ (rounds − 1) × (cost(draft) + cost(review))
```

Listing 4. Example exchange or schema.

Usage records make each term observable per article, per provider and per role.

21 xGovernance

Function	Practice
Govern	named owners for rulebook, facts and approvals
Map	stages, roles and their risks documented
Measure	first-round approval, rounds, stuck items, cost per output
Manage	budgets, limits, human approval, rollback of published items

22 xImplementation roadmap

Phase	Outcome	Activities
1. Discovery	Work broken into stages	map the process; define roles, outputs, approvals and limits
2. Grounding	Rules and facts in one place	write the rulebook; compile verified facts; collect references
3. Pilot	One pipeline live with a person at the gate	maker and checker on two providers; usage logging; a review board
4. Scale	More pipelines, measured	budgets, dashboards of rounds and cost, scheduled runs

23 xGlossary

Maker–checker	One party produces, a different one reviews.
Rulebook	Rules every agent follows, owned by people.
Verified facts	The only claims an agent may make.
Map-reduce	Process parts separately, then combine.
Budget	A hard limit on time, rounds or tokens.

24 References

1. Anthropic. *Claude API documentation: Messages and models*.
2. OpenAI. *API reference: Responses and Chat Completions*.
3. Anthropic. *Prompt caching*, API documentation.
4. JSON Schema. *JSON Schema: Draft 2020-12*.
5. OWASP Foundation. *OWASP Top 10 for Large Language Model Applications*, 2025.
6. NIST. *Artificial Intelligence Risk Management Framework (AI RMF 1.0)*, 2023.

About this report. Prepared by the Cloudnix engineering team from systems we have designed, built and operated. Client and product names are withheld; illustrative values are marked as examples. Cloudnix Software Labs, Bengaluru, India · cloudnix.com. This report may be shared freely in its entirety.

Document control			
Report	CNX-TR-2026-05	Version	1.1
Issued	November 2026	Status	Published
File	cloudnix-agentic-workflows-v1.1.pdf		