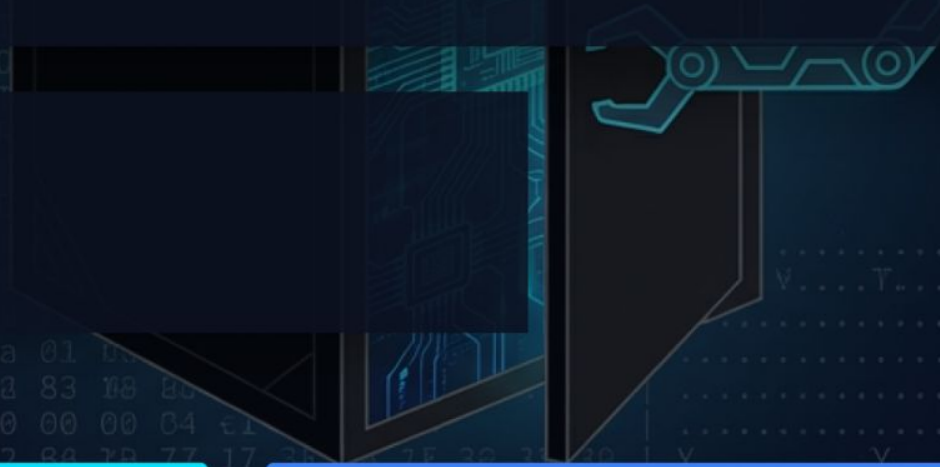
Lessons Learned from Building Tool-Orchestrated LLM Agents for Root Cause Analysis of Large-Scale Logs

Juan Mejia S.
IBM Deutschland R&D GmbH



IN COLLABORATION WITH:

Dipl. Inform. Ruben Nuredini
Heilbronn University of Applied Sciences

Dr. Simon Spinner
IBM Deutschland R&D GmbH

RCA and Mainframe Logs

Defining the problem space and exploring our raw data

Defining RCA

Root Cause Analysis (RCA) is essentially digital forensics. It is the process of tracing a system failure back to its absolute origin.

The Source: IBM Mainframe

We acquire these mainframe logs via two channels: Internal Testing and "Call Home" events.

Discovering the Constraints

Analyzing this data immediately revealed our biggest hurdles: scale and security.

</> parsed_event_schema.json (Preview - MOCK DATA)


```
{
  "title": "log Processor",
  "events": [
    {
      "index": 16127,
      "utcMsec": 1719298246390,
      "severity": "ERROR",
      "num_blocks": 4,
      "blocks": [
        {
          "id": 128,
          "content": [
            "2026-04-08 09:12:33.423 [FATAL] [org.springframework.boot.SpringApplication] Application run failed",
            "2026-04-08 09:12:33.425 [FATAL] [java.lang.OutOfMemoryError] Java heap space: failed reallocation"
          ]
        },
        {...}, {...}, {...}
      ]
    }
  ]
}
```

Constraints & Research Gap

Diagnosing IBM Mainframe logs under strict privacy and infrastructure boundaries

The Mandatory Constraint

No-Cloud Mandate: Strict data privacy rules meant zero external APIs. We could not send this sensitive telemetry to ChatGPT, Gemini, or Claude.

The Weight of the Data

Standard LLMs only support a fraction of this (e.g., a 200k token limit), making direct reading impossible.

- ⚠ Average File: 231MB (~460k lines)
- ⚠ The Extremes: Up to 1.21GB (~11.8M lines)

The Research Gap

No Standard Playbook: When we began, there were no established "Do's and Don'ts" for building autonomous AI agents.

The Technology Stack

Orchestration, Evaluation & Local Models

Core Infrastructure

To meet strict privacy mandates (no cloud AIs) while enabling complex multi-step reasoning, we built a fully local orchestration and evaluation pipeline.

LangGraph

Stateful orchestration layer managing cyclical workflows, tool routing, and memory check-pointing.

Promptfoo

Evaluation framework used to rigorously benchmark LLMs.

Ollama

Local runtime environment hosting quantized open-weight models, ensuring zero data egress.

Constraint Driven

Local-only architecture forced a strict token budget, prioritizing models with strong instruction-following over raw parameter count.

LOCAL MODELS

Qwen 3 (14B)

Used for orchestration and RCA

DeepSeek-R1 (14B)

Use Only for RCA

Granite 3.3 (8B)

Use Only for RCA

**All models quantized and served via Ollama to follow the constraint*

How AI Agents Think

ReAct Loop: Reason + Act

Core Idea

Rather than solving the task in one step, the agent works through an iterative decision loop. Each cycle combines reasoning with external actions.

Typical Loop

Observe
Read current log context, state, and previous tool outputs.

Reason
Decide what information is missing and what should happen next.

Act
Call the most relevant tool (filter logs, inspect metadata, extract lines, etc.).

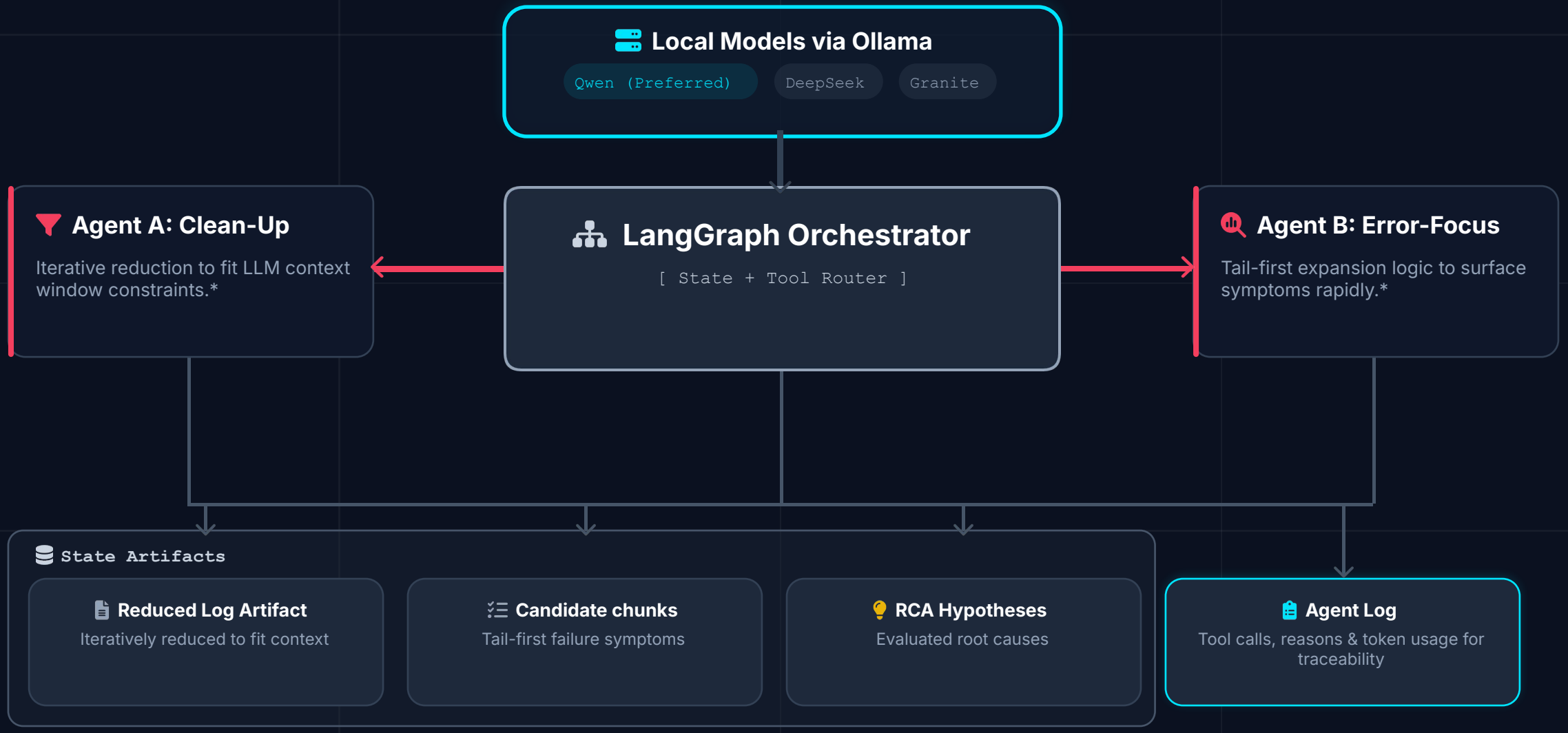
Update
Use the returned result as new context

THE LOOP PENALTY

When any step in the loop fails, the next decision is built on flawed context. Small reasoning errors can compound across multiple iterations.

The Dual-Agent Strategy

LangGraph Orchestration Layer Coordinating Stateful Tasks



** Parallel Approaches*

The "Clean-Up" Workflow

Agent A: Iterative Context Reduction

Primary Goal

Produce a **Reduced Log Artifact** that fits the LLM's limited context window without losing critical causal signals.

Tool Registry (4 Core of 15 Total Tools)

`scan_log_metadata_tool`

Scans overall structure and metadata of the raw log artifact to identify key boundaries.

`time_range_filter_tool`

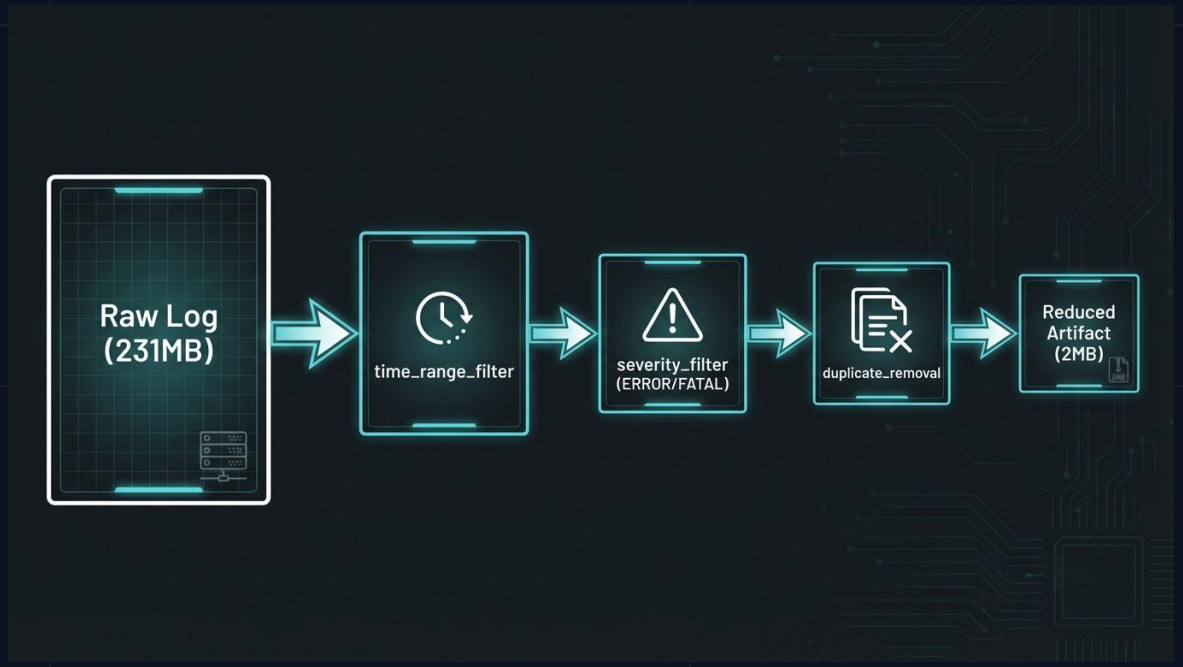
Isolates log lines strictly to the temporal window immediately preceding the failure event.

`create_filtered_log_file_tool`

Generates a newly reduced log artifact based on defined parameters and tool bounds.

`scan_block_stats_tool`

Extracts statistical properties of specific log blocks to guide further reduction.



tool_invocation_example.json

```
tool_calls = [{'name': 'create_filtered_log_file_tool', 'args': {
'block_id_min': 128,
'input_log_key': 'filtered_tail',
'make_active': True,
'min_block_lines': 5,
'output_key': 'final_excerpt'
 }}]
```

The "Error-Focus" Workflow

Agent B: Tail-First Context Expansion

Primary Goal

Start from the most recent failure events and **incrementally explore** backwards. Expand the context window only as needed to form a coherent hypothesis.

Conditional Logic Loop

Start at Tail

Identify the most recent ERROR or FATAL event to use as the pivot point.

Expand Context

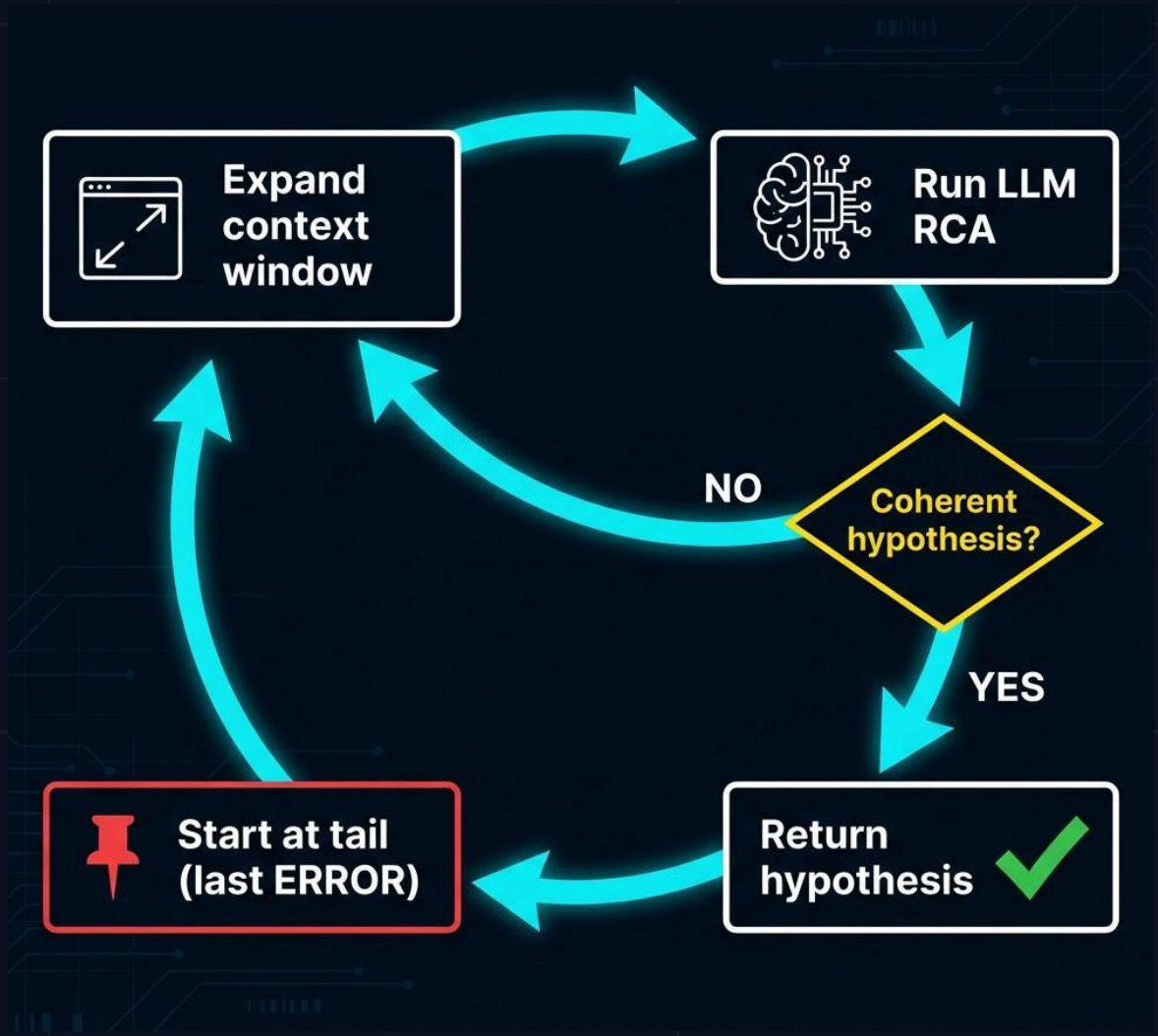
Load a targeted sliding window of log lines surrounding the pivot point.

Run LLM RCA

Instruct the LLM to analyze the isolated window and draft a root-cause hypothesis.

Evaluate & Iterate

If hypothesis is missing evidence, expand window and loop. Stop if coherent.



Accuracy vs. Orchestration

Baseline Performance vs. Agentic Workflows

Baseline: Direct Prompting (~30%)

Feeding heavily truncated log chunks directly to the LLM yielded highest end-to-end accuracy. A simpler context window meant fewer parsing failures and higher reliability.

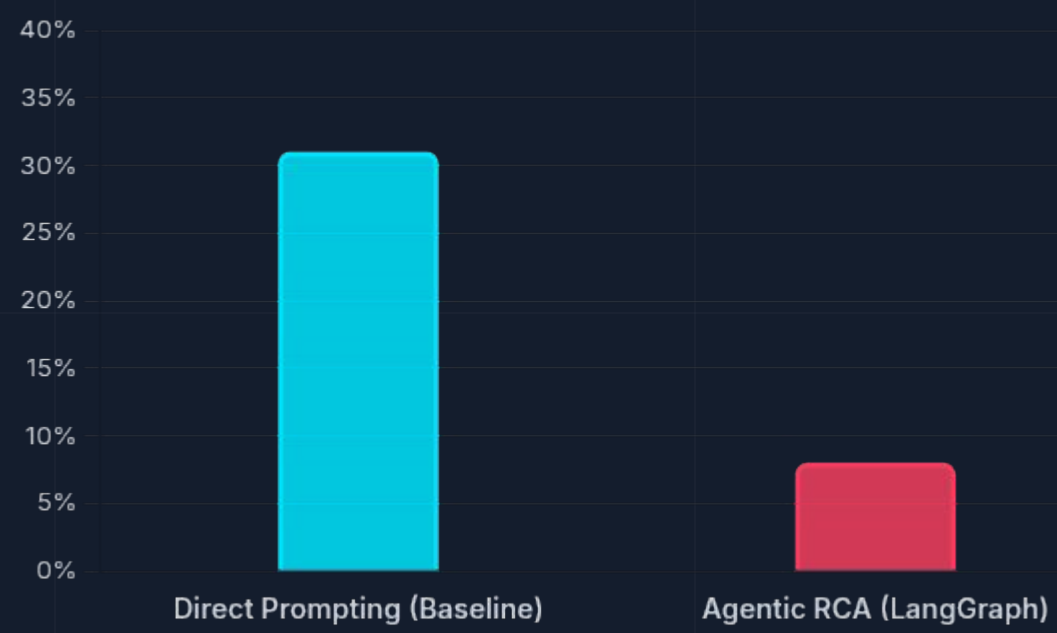
Agentic Workflows (<10%)

The dual-agent LangGraph architecture drastically underperformed expectations. Compounded errors across reasoning steps destroyed accuracy.

- ⚠ Hypothesis generation degraded over loops
- ⚠ Premature conclusions halted investigations

Key Insight

Added orchestration complexity did **NOT** improve accuracy. Instead, it introduced fragile transitions, parsing failures, and state drift.



THE ORCHESTRATION PENALTY

Every tool call is an opportunity for the LLM to hallucinate or drop context. Multi-step reasoning essentially multiplies the failure rate of the underlying model.

Tool-Calling Realities

Valid JSON Schema Adherence

The Winner: Qwen 14B

Only local model capable of **consistent JSON tool-calling** in Ollama. Maintained stable schemas and respected strict invocation contracts across complex contexts.

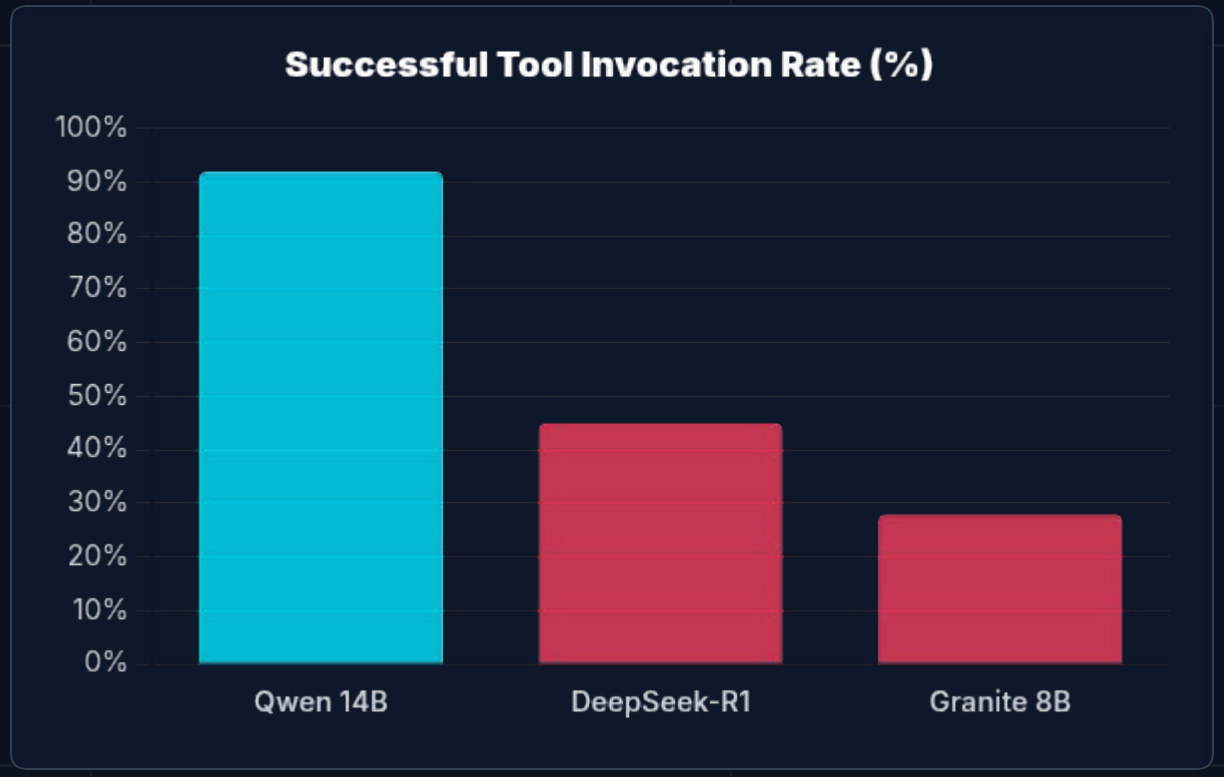
The Failures

DeepSeek-R1 (Ollama) & Granite 8B: Struggled heavily with tool-invocation logic.

- Contract drift and malformed JSON
- Silent loop exits instead of explicit calls

Operational Impact

Inconsistent formats led directly to **retry storms**, causing severe state divergence in the LangGraph checkpoints and premature graph halts.

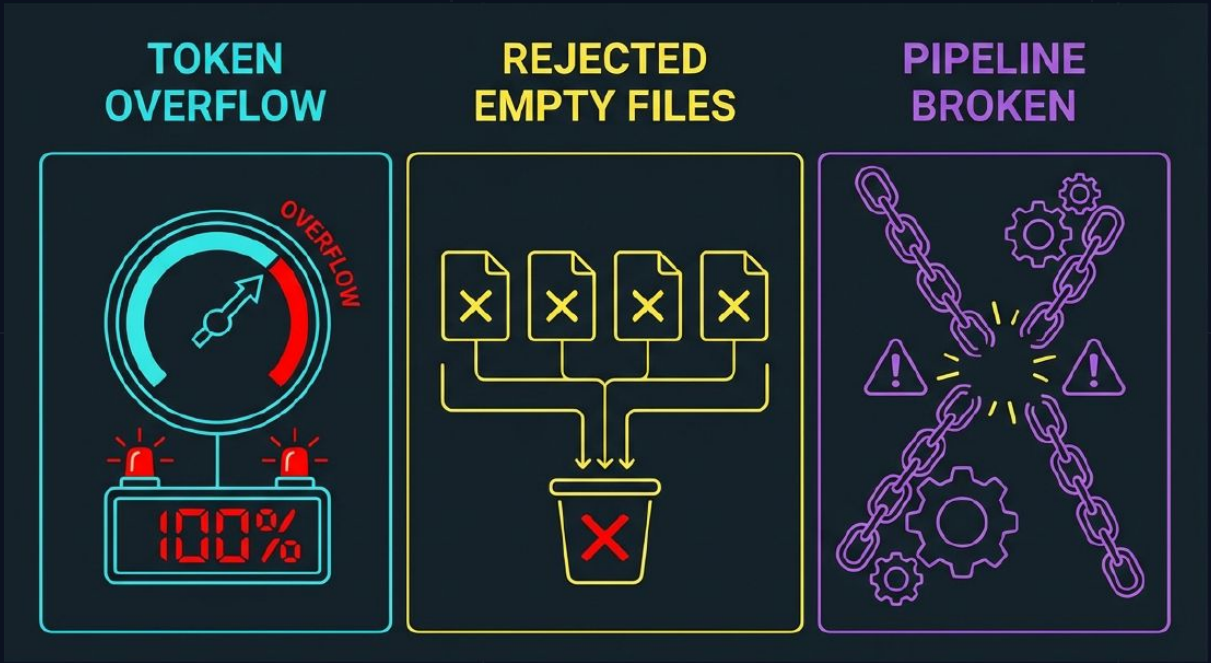


SCHEMA ADHERENCE CRITICALITY

Models that parse intent but fail to structure valid JSON inputs break deterministic graph routing immediately.

Clean-Up Agent Failures

Agent A: Iterative Reduction Pitfalls



Token Issues

Agent completes reduction early, ignoring token budget constraints and overflowing the context window.

Over-Filtering

Model repeatedly tries to create empty files despite tool errors preventing destructive actions.

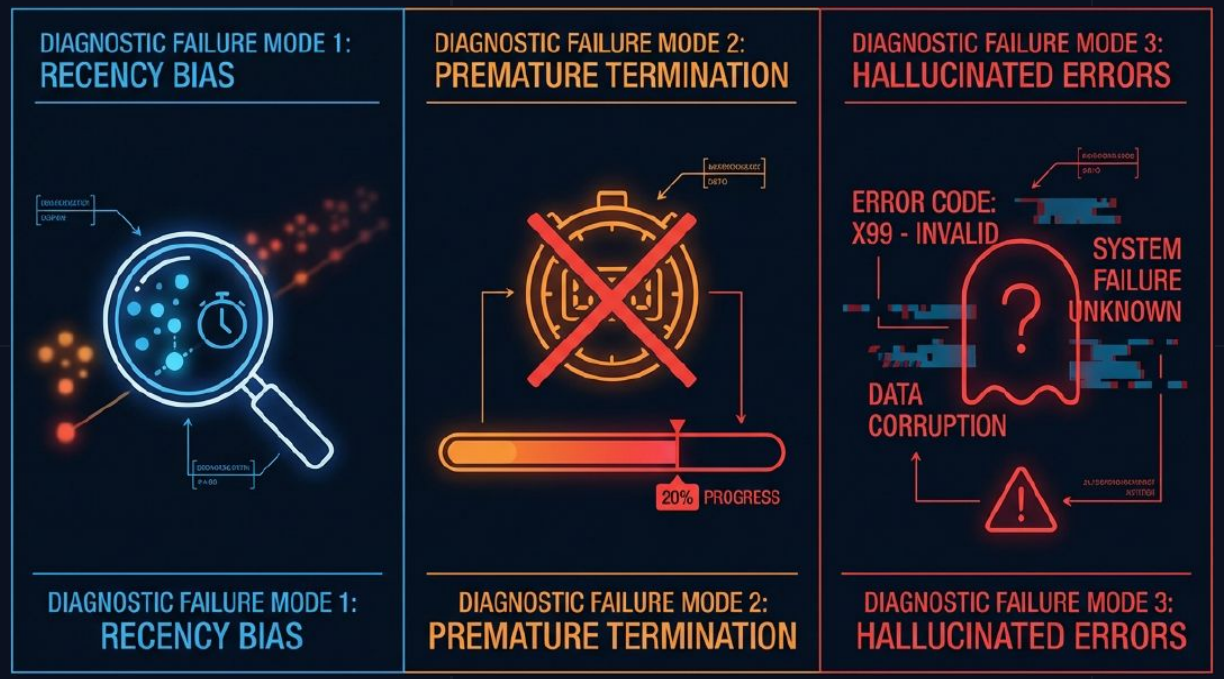
Pipeline Break

Repeated tool failures induce amnesia; the agent forgets its task and halts execution entirely.

Key Takeaway: Providing an LLM with powerful filtering tools does not guarantee optimal resource usage. Without hard algorithmic guards to enforce budgets and break bad loops, autonomous agents easily spiral into destructive or incomplete states.

Error-Focus Agent Failures

Agent B: Diagnostic Blind Spots



Recency Bias

Agent over-indexes on recent events, missing the broader causal chain completely.

Premature Stopping

Analysis halts after the first log segment, failing to investigate earlier critical symptoms.

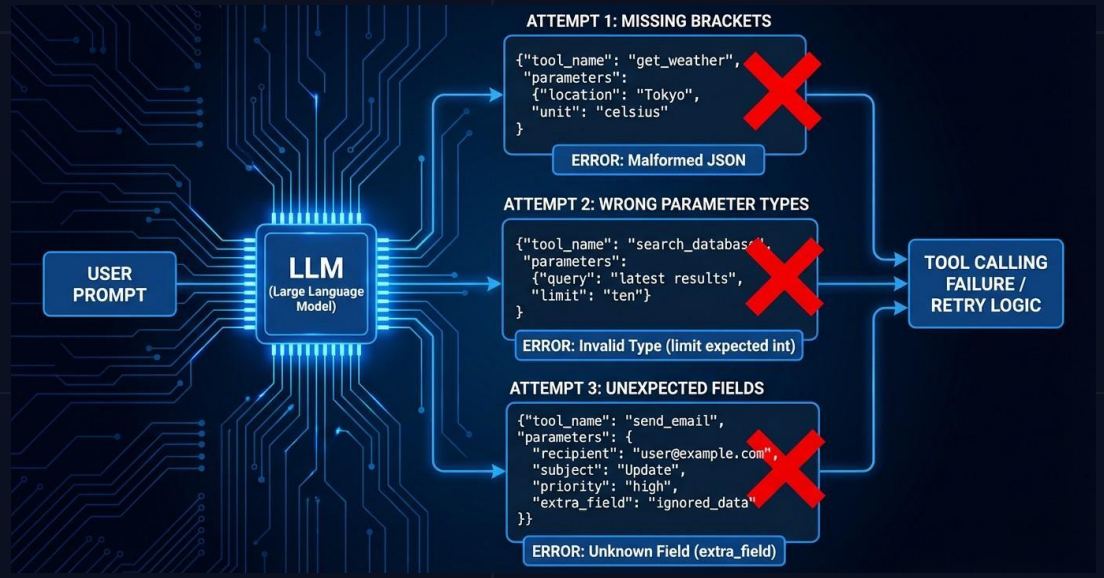
Hallucinations

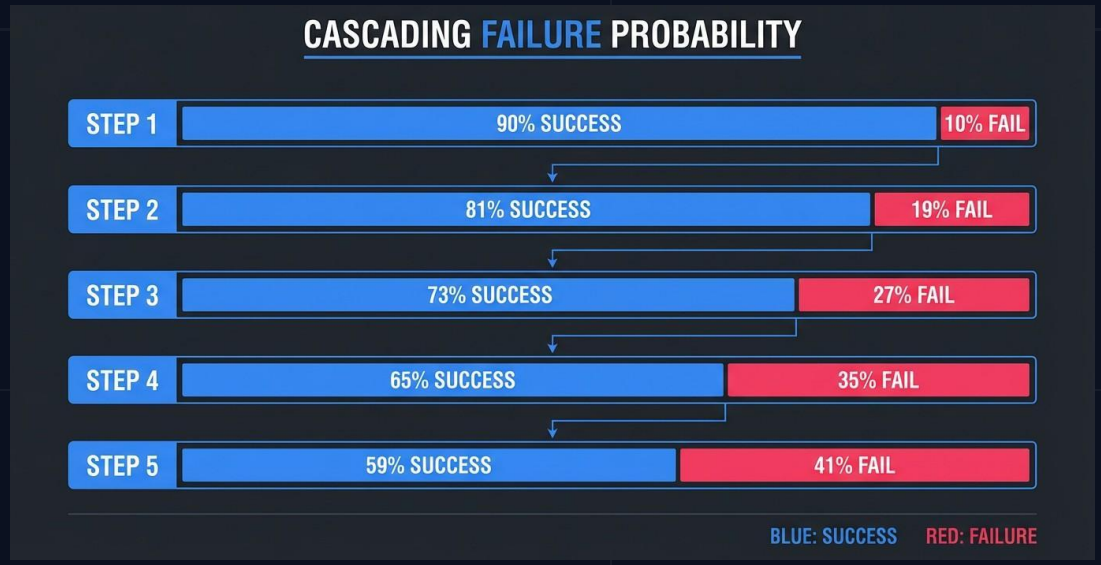
Orchestrator blindly accepts fake RCA errors without validating against the original source text.

Key Takeaway: Diagnostic agents are prone to "satisficing", accepting the first highly confident answer they generate. Robust agentic RCA requires built-in skepticism loops and independent verification tools to combat hallucinated final answers.

System-Level Failures

Orchestration & Infrastructure Pitfalls





Tool Calling Unreliable

Local models struggle with complex JSON schemas, causing drift and premature loop exits.

Orchestration Penalty

90% per-step success = 59% overall completion chance for a 5-step graph.

Key Takeaway: Orchestration overhead often negates the benefits of agentic workflows. Relying on complex, multi-step LLM reasoning loops introduces compounding fragility into the system.

The Autonomy Trap

The Cost of Mundane Decisions

The Illusion of Autonomy

Believing that an AI can natively structure and manage its own complex data pipelines without explicit hard-coded constraints.

The Delegation Error

Treating the LLM as a general-purpose system controller rather than a specialized diagnostic engine.

Triggering the Cascade Penalty

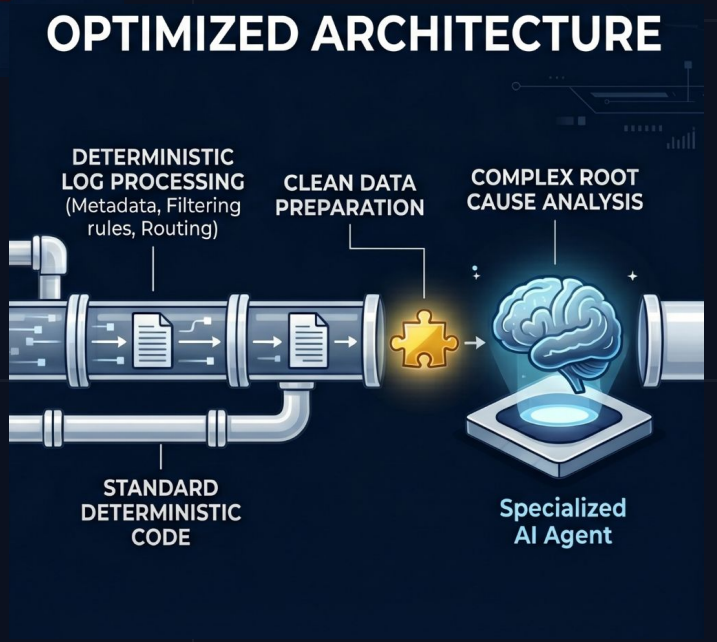
Using probabilistic models for routine tasks turns 1-step scripts into fragile, multi-step loops, compounding system errors

Compounded Fragility

Each mundane decision forces a full reasoning and JSON parsing cycle, drastically increasing the chance of state drift or tool failure

Architectural Misalignment

The architecture failed by giving the LLM too much freedom. Data preparation should be handled by rigid, hard-coded logic, reserving the LLM only for the final root-cause analysis



Can These Failures Be Reduced?

Architectural Optimization & Engineering Mitigations

Note on Residual Risk: Engineering mitigations are layers of defense, like slices of Swiss cheese. While they catch the majority of errors, if the 'holes' (edge cases) align, the system can still fail. Human oversight remains the final, non-negotiable guardrail.

Deterministic Data Pipelines

Replacing LLM-driven filtering, routing, and mundane decisions with rigid, standard code to completely bypass the orchestration penalty.

Agent Specialization

Limiting the AI's autonomy so it only receives clean, pre-processed data, reserving its reasoning exclusively for complex root-cause analysis rather than data prep

Hard Algorithmic Guards

Implementing budget-aware tool wrappers and pre-emptive token truncation to protect the context window.

Stateful Reflection Loops

Validator nodes in LangGraph. Evidence-based hypothesis checking.

Design Recommendations

Pragmatic Rules for Agentic RCA in Production

1 ReAct might not be the answer

- Shift to Stateful Graphs
- Separate Planning & Execution
- Tailor the Architecture

2 Design Tools Like APIs for Juniors

- Keep tools small & single-purpose
- Avoid ambiguity in parameters
- Add examples in tool descriptions

3 Logging & Observability Are Critical

- Log every tool call
- Log every intermediate reasoning step
- Log errors and retries

4 Every Step Adds Failure Probability

- Minimize number of steps
- Prefer simpler pipelines
- Only add steps if they add clear value

⚠ **THE CORE TRADE-OFF:** Agentic autonomy introduces compounding fragility. The most resilient production systems heavily constrain the LLM, treating it as a reasoning engine within a tightly controlled, highly observable pipeline.

The Path Forward & Q&A

Reality Check on Agentic RCA

We haven't solved RCA yet, but we've learned a lot.

Questions?

Local-only (Ollama) + LangGraph traces =privacy-preserving, auditable workflows

RCA and Mainframe Logs

Defining the problem space and exploring our raw data

How do we define the forensic environment and our primary sources of truth?

Defining RCA

Root Cause Analysis (RCA) is essentially digital forensics. It is the process of tracing a system failure back to its absolute origin

The Source: IBM Mainframe

We didn't have to guess why the system broke. The logs explicitly revealed the loop penalties and exactly where the LLM was getting overwhelmed by mundane tasks

The Ultimate Guardrail

When deterministic mitigations inevitably fail, this transparency is our safety net.

The Real Deliverable

Reproducible traces > black-box answers. The value lies in the structured, evidenced path the agent took.