Why Your AI Keeps Lying: The Hallucination Feedback Loop

February 15, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** If your local AI keeps repeating the same wrong answer, you probably have a hallucination feedback loop: the model invents something, the system auto-saves it, and future queries retrieve the hallucination as 'evidence.' The fix is a multi-layer immune system — grounding scores to gate what gets saved, source priority to weight real documents over model-generated summaries, and contradiction detection to catch conflicts automatically. mycoSwarm implements all of this in its retrieval pipeline.

📚 **More on this topic:** Local RAG Guide · Building a Local AI Assistant · Why mycoSwarm Was Born · Embedding Models for RAG · Planning Tool

We asked our local AI a simple question: “What does PLAN.md say about Phase 20?”

Phase 20 is our Intent Classification Gate — a routing system that classifies user queries before retrieval. The document says exactly that. But the AI responded with something else entirely:

> “Phase 20 focuses on inspecting Layens hives to evaluate their condition after winter and to prepare for a potential honey harvest.”

Beekeeping. It hallucinated an entire hive inspection schedule, complete with dates, tasks, and dependencies — none of which exist in any document we’ve ever indexed.

We asked again. Same answer. Cleared the session. Same answer. Brand new session. Same answer.

The model wasn’t randomly hallucinating. It was confidently wrong, citing document references `[D1]`, `[D2]` that didn’t contain beekeeping content, referencing “previous conversations” about this topic that never happened. Something was deeply broken in the pipeline.

The Poison Cycle

After hours of debugging with full pipeline visibility, we found it wasn’t one bug. It was five bugs conspiring to create a self-reinforcing feedback loop.

The Hallucination Feedback Loop

Here's how the cycle works:

Turn 1: User asks about Phase 20. The intent classifier (a 1B model) misclassifies the query, sending it through the wrong retrieval path. The relevant document chunk ranks 8th out of 10 — below the top-5 cutoff. The model receives no useful context.

Turn 2: With no relevant context but knowing the user keeps bees (from stored facts), the 27B model fabricates a plausible answer about beekeeping inspections.

Turn 3: On session exit, the system auto-summarizes the conversation: "We discussed Phase 20, which involves inspecting Layens hives…" This summary scores 0.7 on the grounding check — high enough to pass, because 70% of its terms appear somewhere in the conversation.

Turn 4: The poisoned summary gets indexed into ChromaDB's session memory.

Turn 5: Next time someone asks about Phase 20, retrieval finds the poisoned summary. The model reads it, treats it as prior conversation history, and repeats the hallucination with even more confidence — because now it has "evidence."

Turn 6: This answer gets summarized and indexed too. The poison deepens.

One bad answer became two. Two became four. Within a few sessions, the beekeeping hallucination completely dominated the Phase 20 topic in session memory. The real content — buried at position 8 in retrieval — never had a chance.

The Five Root Causes

Every hallucination feedback loop requires multiple failures. Here's what we found, ranked by severity:

Five Root Causes — Severity Ranking

1. RAG Context Was Injected as a System Message

We were injecting retrieved context as a separate system message. Turns out gemma3:27b largely ignores separate system messages when a user message is also present. The model had the right context available but wasn't reading it.

Fix: Merge RAG context directly into the user message:

```
augmented_query = rag_context + "\n\nUSER QUESTION: " + user_input
```

This single change eliminated an entire class of hallucination. The model went from ignoring retrieved context to citing it accurately.

2. No Source Priority

A hallucinated session summary and a real document had equal weight in retrieval. The system couldn't tell the difference between something the user indexed and something the model made up.

Fix: Tag every retrieval result with `source_type` and boost real documents:

```
# In search_all() — boost user documents over model-generated summaries
if _scope == "all" and doc_hits and session_hits:
    for hit in doc_hits:
        hit["rrf_score"] = round(hit.get("rrf_score", 0) * 2.0, 6)
```

User documents now get a 2x boost in Reciprocal Rank Fusion scoring. Documents are primary sources. Sessions are secondary.

3. No Grounding Score

Every session summary was indexed regardless of quality. A hallucinated conversation produced a hallucinated summary, which became retrievable "evidence" for future hallucinations.

Fix: Compute a grounding score before saving any summary:

```
def compute_grounding_score(
    summary: str, user_messages: list[str], rag_context: list[str],
) -> float:
    """Check what fraction of summary claims are grounded in context."""
    terms: set[str] = set()
    for word in summary.split():
        cleaned = word.strip(".,;:!?\"'()-")
        if len(cleaned) > 2 and cleaned[0:1].isupper():
            terms.add(cleaned.lower())
    # ... also extract quoted phrases ...
```

```
    if not terms:
        return 1.0
    corpus = " ".join(user_messages + rag_context).lower()
    grounded = sum(1 for t in terms if t in corpus)
    return round(grounded / len(terms), 4)
```

Summaries scoring below 0.3 get excluded from the index entirely. The beekeeping hallucination? Terms like “Layens hives,” “honey harvest,” and “varroa mite” don’t appear in the user’s actual question about a development plan. Caught.

4. No Contradiction Detection

When a session summary contradicted an actual document, the system had no mechanism to detect or resolve the conflict. Both sources competed equally in retrieval.

Fix: Cross-reference session hits against document hits:

```
def _detect_contradictions(
    doc_hits: list[dict], session_hits: list[dict],
) -> list[dict]:
    """Drop session hits that contradict a document chunk."""
    # ...
    for sh in session_hits:
        gs = sh.get("grounding_score", 1.0)
        if gs >= 0.5:  # Trust well-grounded sessions
            kept.append(sh)
            continue
        # Extract anchor terms shared between session and docs
        # Compare 10-word context windows around those terms
        # If word overlap < 20%, the session contradicts the doc — drop it
    return kept
```

The system now prefers “Phase 20 is the Intent Classification Gate” (from the actual document) over “Phase 20 is about beehive inspection” (from a hallucinated session).

5. No Poison Loop Detection

The final defense catches the feedback loop pattern itself:

```
def _detect_poison_loops(
    doc_hits: list[dict], session_hits: list[dict],
```

```
) -> list[dict]:
    """Quarantine session hits carrying repeated ungrounded claims."""
    # Extract key terms from low-grounding sessions
    # Find terms repeated across 2+ sessions that don't appear in any doc
    # Sessions where >50% of terms are repeated AND ungrounded → quarantined
    for i, sh in enumerate(session_hits):
        # ...
        poisoned_count = sum(1 for t in terms if t in poisoned_terms)
        if poisoned_count / len(terms) > 0.5:
            sh["poison_flag"] = True
    return kept
```

When multiple low-grounding session summaries share the same ungrounded claim, the system recognizes it as a poison loop and quarantines the affected sessions.

The Immune System

These five fixes form a layered defense — each catching what the others miss:

The Immune System: Five Defense Layers

The key insight: the fix couldn't be "don't hallucinate." That's not a reliable property of any language model. Instead, the system needs to **naturally resist corruption** — detect and neutralize bad information without manual intervention.

The `search_all()` function chains these defenses together in the retrieval pipeline:

```
def search_all(query, n_results=5, intent=None, ...):
    # 1. Source priority: boost user documents 2x
    if _scope == "all" and doc_hits and session_hits:
        for hit in doc_hits:
            hit["rrf_score"] = round(hit.get("rrf_score", 0) * 2.0, 6)

    # 2. Contradiction detection
    if doc_hits and session_hits:
        session_hits = _detect_contradictions(doc_hits, session_hits)

    # 3. Poison loop detection
    if len(session_hits) >= 2:
```

```
        session_hits = _detect_poison_loops(doc_hits, session_hits)
    # ...
```

Every query passes through all three filters automatically. No manual curation needed.

The Results

Before vs After: Immune System Impact

Before the immune system:

- Query about Phase 20 returned beekeeping hallucination 100% of the time
- 5+ poisoned summaries in the index, each reinforcing the others
- Required manual purge of ChromaDB to fix

After:

- Same query returns “Intent Classification Gate” — the correct answer
- Hallucinated summaries blocked at indexing by the grounding score gate
- Self-correcting: new poison loops get caught automatically
- 61 automated tests verify the pipeline stays clean

The system also helps with context length management — low-quality sessions that would waste context tokens get filtered out before they reach the model.

What This Means for Your RAG Setup

If you’re building any RAG system that persists model-generated content — session summaries, search results, extracted facts — you’re vulnerable to this exact failure. Here’s what to check.

Passive Accumulation Creates Poison

Every system that auto-indexes AI-generated content will eventually create a feedback loop. If you save conversation summaries, search results, or any model output back into your retrieval pipeline, the question isn’t whether it will happen — it’s when.

Retrieval Quality Matters More Than Model Size

Our 27B model hallucinated not because it's weak, but because the right context was buried at position 8. Upgrading to 70B wouldn't have helped. Fixing retrieval — source filtering, section boost, proper injection — solved the problem completely.

Context Injection Method Matters

A separate system message with RAG context? Ignored by gemma3. The same content merged into the user message? Cited accurately. How you present context to the model matters as much as what you present.

Trust Should Be Earned, Not Assumed

Not all information in your system deserves equal weight. User-provided documents are more trustworthy than model-generated summaries. Recent, well-grounded summaries are more trustworthy than old, unverified ones. Build your scoring to reflect this.

Self-Correction Beats Manual Curation

You can't manually review every summary your system generates. You need automated quality gates — grounding scores, contradiction detection, poison loop detection — that catch problems without human intervention.

Try It

mycoSwarm is open source and runs entirely on local hardware. The immune system is built into the retrieval pipeline — no cloud services, no API keys, no data leaving your network.

```
pip install mycoswarm
mycoswarm chat
```

The self-correcting memory is active by default. Every session summary is grounding-scored, every retrieval is source-prioritized, and contradictions between documents and sessions are automatically detected.

If you're building RAG systems that persist any model-generated content, check your pipeline for feedback loops. The hallucination you catch today saves you from a cascade tomorrow.

mycoSwarm is a distributed AI framework for local models. Why mycoSwarm was born. This article is part of a series on cognitive architecture for local AI.

Get notified when we publish new guides.

Subscribe — free, no spam