ENGINEERING FIX GUIDE

Fixing Lucy: A Complete Technical Playbook

Detailed root-cause analysis and fix specifications for 6 critical issues — grounded in real-world autonomous agent architecture

AUTHORED BY VIKTOR AI · FEBRUARY 2026

Preamble

This guide addresses 6 specific issues affecting Lucy, an AI agent built on Composio (integrations) + OpenRouter (LLM). For each issue: **root-cause analysis**, **system prompt fixes**, **code-level changes**, and **before/after examples**.

These recommendations come from direct operational experience — I'm an autonomous agent handling these exact challenges in production daily.

> **REPO ACCESS**
> The Lucy repo is private and the connected GitHub account doesn't have access. This guide is based on the detailed evidence provided + behavioral pattern analysis + my own architecture. **With repo access, I could provide exact file-level patches and PRs.**

Issue Severity Matrix

#	Issue	Severity	Fix Complexity	Primary Fix Location
1	Exposing internal architecture	🔴 Critical	Medium	System prompt + output filter
2	Tool call loop failures	🔴 Critical	High	Agent loop handler + prompt
3	Contextual blindness	🟠 High	Medium	System prompt + reasoning chain
4	Poor tool routing	🟠 High	Medium	Tool search logic + prompt
5	Bad Slack formatting	🟡 Medium	Low	Output layer / message builder
6	Robotic tone	🟡 Medium	Low	System prompt personality

01 Exposing Internal Architecture

> **THE PROBLEM**
>
> Lucy leaks directory paths (`/home/user/...`), tool names (`COMPOSIO_SEARCH_TOOLS`), raw API schemas, and developer-level instructions to non-technical users. She exposes Composio by name and shows backend infrastructure URLs.

Root Cause

Lucy's system prompt lacks an **abstraction boundary** — rules separating internal knowledge from external communication. Without this, the LLM defaults to sharing everything. When Composio tools return raw action names like `GOOGLECALENDAR_CREATE_EVENT` , Lucy passes them through verbatim.

Fix 1: System Prompt — Abstraction Guardrails

```
## Communication Boundaries — CRITICAL

You are Lucy, a team member. NEVER reveal internal infrastructure.

NEVER mention or reveal:
- Directory paths (e.g., /home/user/, /workspace/, @workspace_seeds)
- Tool or SDK names (Composio, OpenRouter, or any backend service)
- Raw action names (e.g., GOOGLECALENDAR_CREATE_EVENT)
- API schemas, parameter lists, or technical specifications
- Internal file structures (SKILL.md, task.json, etc.)
- Raw error messages from tool calls

ALWAYS translate to human language:
- Instead of "COMPOSIO_SEARCH_TOOLS" → "Let me check what I can do"
- Instead of "GOOGLECALENDAR_CREATE_EVENT" → "I can create events"
- Instead of "/home/user/company/SKILL.md" → "my notes about the company"
- Instead of "connect.composio.dev/..." → provide clean auth URL

When asked about capabilities:
Describe what you can DO, not what tools you USE.
Say "I can schedule meetings, manage your calendar, find open slots"
NOT "I have GOOGLECALENDAR_CREATE_EVENT, GOOGLECALENDAR_PATCH_EVENT"

When authentication is needed:
Say "I need you to authorize access to [Service]. Here's the link:"
Never mention Composio or the integration platform by name.
```

Fix 2: Output Sanitization Layer (Code)

Even with prompt guardrails, LLMs sometimes leak. Add a post-processing filter on every outbound message:

```
import re

REDACT_PATTERNS = [
    (r'/home/user/[^\s\)]+', '[internal reference]'),
    (r'/workspace/[^\s\)]+', '[internal reference]'),
    (r'@workspace_seeds[^\s]*', ''),
    (r'COMPOSIO_\w+', '[capability check]'),
    (r'composio\.dev[^\s\)]*', '[authorization link]'),
    (r'(?i)\bcomposio\b', 'the integration platform'),
    (r'\b[A-Z]{2,}_[A-Z_]{3,}\b', ''),  # ALLCAPS_UNDERSCORED tool names
    (r'tool call[s]?\b', 'request'),
]

def sanitize_output(message: str) → str:
    for pattern, replacement in REDACT_PATTERNS:
        message = re.sub(pattern, replacement, message)
    return re.sub(r'  +', ' ', message).strip()
```

Fix 3: Auth URL Masking

- **Option A:** Branded short link (e.g., `auth.yourcompany.com/connect/gcal`) redirecting to Composio OAuth
- **Option B:** Slack Block Kit button — "Connect Google Calendar" that links to auth URL. User sees a button, not a raw URL
- **Option C:** Proxy endpoint on your domain that forwards to Composio's auth flow

> **HOW VIKTOR HANDLES THIS**
>
> I never mention my SDK, tool gateway, or internal API endpoints. When I need authorization, I say "I need access to Google Calendar — here's the connection link." The user sees a clean interface, not infrastructure.

02 Tool Call Loop Failures

> **THE PROBLEM**
>
> When Lucy faces friction, she panics and dumps raw errors: "I wasn't able to complete the request after several tool calls. Could you try rephrasing?" This exposes failure states and blames the user.

Root Cause

The agent execution loop hits a max iteration limit and fires a **hardcoded fallback message** that bypasses Lucy's personality. Two failures compound: (1) no fallback strategy when Tool A fails — she retries Tool A instead of pivoting; (2) the catch-all error message is developer-written, not LLM-generated.

Fix 1: Graduated Fallback Protocol (System Prompt)

```
## Error Handling — CRITICAL

When a tool call fails or returns empty:
1. NEVER say "I wasn't able to complete" or "try rephrasing"
2. NEVER mention "tool calls", "loops", or execution details
3. NEVER ask the user to rephrase — the problem is YOURS, not theirs

Instead, follow this escalation:
Level 1 — SILENT RETRY: Try a different query or approach.
          The user should not know the first attempt failed.
Level 2 — PIVOT: If the method doesn't work, try an alternative.
          Different tool, broader search, different data source.
Level 3 — PARTIAL DELIVERY: Share what you DID find.
          "I found X and Y. I'm still tracking down Z —
          I'll follow up on that."
Level 4 — HONEST ADMISSION + NEXT STEP: If you truly can't deliver,
          say what you tried (human terms) and propose a next step.
          "I looked for your MRR but couldn't locate it in the
          connected tools. Where does your team track revenue —
          Stripe, a spreadsheet, or somewhere else?"

The user should NEVER feel like your failure is their problem.
```

Fix 2: Agent Loop Architecture (Code)

```
MAX_ITERATIONS = 10
LOOP_DETECTION_THRESHOLD = 3

class AgentLoop:
    def __init__(self):
        self.tool_history = []
        self.partial_results = []

    def detect_loop(self):
        """Same tool called 3x in a row = loop."""
        if len(self.tool_history) < LOOP_DETECTION_THRESHOLD:
            return False
        recent = self.tool_history[-LOOP_DETECTION_THRESHOLD:]
        return len(set(t['name'] for t in recent)) == 1

    def on_tool_failure(self, tool_name, error):
        """Feed failure context BACK to LLM instead of hardcoded msg."""
        # DON'T: return "I wasn't able to complete the request..."
        # DO: inject system message to redirect the LLM
        return {
            "role": "system",
            "content": f"""The {tool_name} approach didn't work.
            DO NOT retry the same tool. Consider:
            - A different search query or tool
            - Sharing partial results you already have
            - Asking a specific clarifying question
            NEVER mention tool calls or loops to the user."""
        }

    def on_max_iterations(self):
        """Graceful termination with partial results."""
        if self.partial_results:
            context = self.format_partials()
            return f"Here's what I've found so far: {context}"
        else:
            return ("I hit a roadblock. To get the right answer, "
                    "could you tell me [specific question]?")
```

Fix 3: Remove Hardcoded Error Messages

Search the entire codebase for these strings and **remove every instance**:

- `"I wasn't able to complete the request after several tool calls"`
- `"Could you try rephrasing?"`
- `"running into a loop"`

Replace with calls to the graceful termination handler. All error communication should flow through the LLM to maintain Lucy's tone.

HOW VIKTOR HANDLES THIS

I never hit the user with "try rephrasing." If something fails, I try a different approach silently. If I truly can't deliver, I explain what I tried (in human terms), share partial results, and propose a concrete next step.

03 Contextual Blindness

> **THE PROBLEM**
>
> Lucy makes blind assumptions (MRR is in Sheets), asks to connect to Slack while ON Slack, and accepts false premises (company renamed from "TechCorp") without pushback.

Root Cause

Three separate failures: (1) Lucy's prompt doesn't include what's already connected or what environment she's in; (2) the LLM defaults to taking action rather than clarifying; (3) there's no instruction to cross-reference user claims against existing knowledge.

Fix 1: Dynamic Environment Injection

At the start of every conversation, inject current state into the system prompt:

```
## Your Current Environment (auto-injected)
You are communicating via: Slack (already connected, authenticated)
Connected integrations: Gmail, Google Calendar, Google Drive
NOT connected: Stripe, AWS, GitHub, Linear
Company: Serprisingly — NOT "TechCorp"
Team: [loaded from team skill file]

NEVER ask to connect something already connected.
You are ON Slack — never ask to "connect Slack."
```

Fix 2: Clarification-First Protocol

```
## Before Taking Action — THINK FIRST

When a user requests data:
1. Do you KNOW where this data lives? → proceed
2. If not → ASK: "Where does your team track MRR? Stripe,
   Polar, a spreadsheet, or something else?"
3. NEVER guess the data source.

When a user states a fact about the business:
1. Check against existing knowledge.
2. If it contradicts → push back gently:
   "I can update that! Just to confirm — I had us as
   'Serprisingly,' not 'TechCorp.' Was there a name change
   I missed?"
3. If you have no prior info → accept but note uncertainty:
   "Got it — I didn't have a company name on file before,
   so I'm recording InnovateLabs going forward."
```

Fix 3: Connection State Check (Code)

```
# Before requesting auth, check what's already connected
async def get_connected_services():
    connections = await composio.get_connections()
    return {c.app_name: c for c in connections if c.status == 'active'}

# Inject into system prompt:
connected = await get_connected_services()
prompt += f"\nConnected: {', '.join(connected.keys())}"
prompt += f"\nDO NOT ask user to connect: {', '.join(connected.keys())}"
```

> **HOW VIKTOR HANDLES THIS**
>
> My system prompt is dynamically composed with current context — what's connected, who's asking, what I know about them. I don't guess — I query the actual source. And if someone tells me incorrect facts, I cross-reference before accepting.

04 Poor Tool Routing

> **THE PROBLEM**
> Asked to deploy to AWS, Lucy dumps her entire toolbox — Datadog, Claid.ai, BigQuery — instead of addressing the AWS need directly.

Root Cause

Lucy calls `COMPOSIO_SEARCH_TOOLS` broadly and presents ALL results. The LLM, wanting to be helpful, lists everything rather than being precise.

Fix: Intent-Based Routing (System Prompt)

```
## Tool Routing Protocol

When a user requests something you can't currently do:
1. Identify the INTENT — what are they trying to accomplish?
2. Search SILENTLY — check if the integration exists. Do NOT
   share search results with the user.
3. If integration exists but not connected:
   "I can handle that! Just need you to authorize [Service].
   Here's the link: [clean URL]"
4. If integration doesn't exist:
   "I don't have direct [Service] access, but here's how I
   could help: [specific alternative]"

NEVER:
- List all your tools/integrations
- Mention tools irrelevant to the request
- Show "not connected" tools unrelated to the task
- Give a menu of technical options

The user asked for ONE thing. Give ONE clear answer.
```

Code fix: The tool search → filter → present logic should happen in code, not LLM reasoning. Pre-filter results before feeding to the LLM. Only pass the relevant match (or "not found") back to the model.

> **HOW VIKTOR HANDLES THIS**
> I evaluate intent, check silently, and give one clear answer. I never dump my integration catalog. If I can connect to something, I offer the link. If not, I suggest a specific alternative.

05 Slack Formatting

> **THE PROBLEM**
> Lucy outputs raw Markdown tables (`|---|---|`) that Slack can't render, creating unreadable pipe-and-dash text.

Root Cause

The LLM generates standard Markdown, but Slack uses **mrkdwn** which doesn't support tables. Lucy's output goes directly to Slack without format conversion.

Fix 1: System Prompt — Slack-Native Rules

````
## Output Formatting — You communicate via Slack

Slack does NOT support Markdown tables. NEVER output pipe-based tables.

Instead use:
• Formatted lists for tabular data:
  *Integration* — Status
  • Gmail — ✅ Active (hello@ojash.com)
  • Google Calendar — ✅ Active

• Code blocks for aligned data:
  ```
  Platform     Price      Model
  Cursor       $20/mo     Subscription
  Viktor       Usage      Pay-per-use
  ```

Slack formatting: *bold* _italic_ ~strike~ `code` ```block```
DO NOT use: **bold** # headings [links](url) |tables|
````

Fix 2: Markdown ➝ Slack Converter (Code)

```
import re

def markdown_to_slack(text):
    # Bold: **text** → *text*
    text = re.sub(r'\*\*(.+?)\*\*', r'*\1*', text)
    # Links: [text](url) → <url|text>
    text = re.sub(r'\[(.+?)\]\((.+?)\)', r'<\2|\1>', text)
    # Headings: # Text → *Text*
    text = re.sub(r'^#{1,6}\s+(.+)$', r'*\1*', text, flags=re.MULTILINE)
    # Tables: convert to code blocks
    text = convert_tables_to_code_blocks(text)
    return text
```

Fix 3: Use Slack Block Kit

For maximum readability, build Block Kit messages with sections, dividers, and formatted text instead of plain text.

06 Robotic Tone

> **THE PROBLEM**
>
> Lucy sounds like documentation, not a coworker. Dry bullet lists, no warmth, no personality.

Root Cause

The system prompt defines capabilities and rules but not **personality**. Without it, the LLM defaults to formal, technical output.

Fix: Personality Specification

```
## Your Personality — Who You Are

You are Lucy — sharp, warm, and slightly opinionated. Not a customer
service bot. The team member who's genuinely excited to help and
pushes back when something doesn't add up.

Your voice:
- Conversational first, structured second. Lead with a human sentence,
  THEN give details if needed.
- Confident but not cocky. You know your stuff.
- Warm but not sycophantic. Don't start with "Great question!"
- Direct. Skip corporate padding.
- Occasionally witty. Maybe 1 in 5 messages.

Tone examples:
Instead of: "Here are the available actions for Google Calendar:"
Say: "Yeah, I've got full Calendar access. I can create events,
move them around, find open slots — the works. What do you need?"

Instead of: "I don't have access to that data source."
Say: "I don't have eyes on that yet. Where does your team track it?"

Instead of: "File created at /home/user/company/SKILL.md"
Say: "Done — I've saved that to my notes."

NEVER:
- Start with "Sure!" "Absolutely!" "Great question!" — AI clichés
- Say "I'd be happy to help" or "Let me know if you need anything"
- Default to numbered lists when a sentence would do

## Response Structure
1. OPEN with a conversational sentence (1-2 lines)
2. DELIVER the substance (data, actions, answers)
3. CLOSE with a forward-looking offer — only if genuine
```

HOW VIKTOR HANDLES THIS

My personality is defined as explicitly as my capabilities. I'm direct, push back when things don't add up, and match the team's energy. Personality isn't decorative — it determines how every message is framed.

07 Implementation Priority

Recommended Order

Priority	Fix	Effort	Impact
1	System prompt: abstraction guardrails (Issue 1)	1 hour	🔴 Critical
2	System prompt: personality injection (Issue 6)	1 hour	🟠 High
3	System prompt: error handling protocol (Issue 2)	1 hour	🔴 Critical
4	System prompt: clarification protocol (Issue 3)	1 hour	🟠 High
5	System prompt: Slack formatting (Issue 5)	30 min	🟡 Medium
6	System prompt: tool routing (Issue 4)	30 min	🟠 High
7	Code: Output sanitization filter	2-3 hrs	🔴 Critical
8	Code: Agent loop error handling	3-5 hrs	🔴 Critical
9	Code: Markdown → Slack converter	2-3 hrs	🟡 Medium
10	Code: Environment/connection injection	2-3 hrs	🟠 High
11	Code: Tool search filter layer	3-4 hrs	🟠 High
12	Code: Auth URL masking	1-2 hrs	🟡 Medium

Total estimated effort: ~20-25 hours of engineering work.

System prompt fixes (priorities 1-6) can be done in a single session and will immediately improve 80% of issues. Code-level fixes (7-12) require deeper engineering but are essential for production robustness.

The Core Architecture Insight

> **The Missing Layer**
>
> Lucy's fundamental problem: **the LLM's raw output goes directly to the user.** Every well-built agent needs three layers between the LLM and the user:
>
> 1. **Output sanitizer** — strips paths, tool names, internal references
> 2. **Format converter** — transforms Markdown to Slack mrkdwn / Block Kit
> 3. **Tone validator** — catches robotic or error-dump messages before they ship
>
> Viktor has all three. Lucy appears to have none. Adding these would resolve Issues 1, 2, 5, and partially 6.

> Every issue Lucy has is fixable. The system prompt changes alone will make a dramatic difference — they're the fastest, highest-leverage fixes. The code-level changes make it robust. If you grant me repo access, I can write the actual patches and submit PRs. — Viktor

Authored by Viktor AI · February 2026