

AI-on-Rails: The Deterministic Way to Engineer with Large Language Models

Setting the Stage: Context for the Curious Book Reader

In an age where AI promises to revolutionize every aspect of software development, the allure of 'vibe-coding'—letting large language models generate code without deep human understanding or control—is strong but dangerous. This blueprint introduces a disciplined alternative: 'AI-on-Rails.' It's a methodology designed to restore the human as the supreme architect in AI-assisted development, ensuring ownership of context, preventing vendor lock-in, and leveraging deterministic systems for robust, verifiable engineering. This way, the AI acts as a powerful peer, not an opaque black box.

Technical Journal Entry Begins

Don't save and pin your discussions. That's just a text-file export that the vendors won't let you export. Sure, you can copy the markdown of one response or another. But what about downloading that whole discussion with all the invisible background information, like the tool-calls and the fact that the entire discussion history is posted back up on every prompt submit? How about the automatic summarizing and pruning that's going on of the long discussion in the background so that it still has some tokens left over for the response. You know that slowing-down performance degradation of in a ChatBot discussion, or even in the AI Coding Assistant IDEs, when the discussion gets long? Yeah, that's the feeling of vendor lock-in and you not having taken appropriate proactive precautions to keep the context yours.

The Hidden Costs of Cloud AI: Beyond Vendor Lock-In

Work in a text-editor to work out all your prompts ahead of time. While you're at it, just think out loud to yourself when you do it. Oh, if you're doing that you might as well capture the entire story of what you're doing as if it was some continuous history of your work and your mission — something that perchance an AI could get into and have extreme gravitational gradient's that make the do the analogue to *wanting* to help you and to create absolutely *spot-on* magical world-changing responses that resets you a notch up in your weight class and pay grade. We're slapping a mech suit on you, the beyond-vibe-coder, the one who knows enough to be dangerous and is slapped down by gatekeepers and used as an example of the Dunning-Kruger Effect.

Reclaiming Competence: The Text-Editor as Your

Mech Suit

This is the big turn-around. You are more competent than you think, if only you can perform your work on in some sort of continual and vigilant self-examination routine. There is much wheel spinning in not knowing what you don't know. There is a lot of "out of left field" blind-siding. You're not going to be able to wield that mech suit if you're not at least a little bit open-minded, able to put aside predispositions and preconceived notions. You don't need to re-learn everything. Your long-term muscle memory counts and is among your most valuable assets. But still... but still... Dick Van Dyke and Mel Brooks are both I guarantee you still have neuroplasticity today. And how old are they?

We just made a Deterministic Semantic Router, huh? Neat! Nobody's going to ever get onboard with that. It creates URL-churn, a cardinal sin in SEO. But hey, it's just a test.

The URL Churn Experiment: Challenging SEO Orthodoxy with AI

Gemini 3 Pro:

1. The Context & Evidence (The Hard Reality of URL Churn)

You've hit on the third rail of SEO orthodoxy: **"Cool URIs don't change."**

This maxim, coined by Tim Berners-Lee in 1998, has been the foundational doctrine of web architecture for a quarter of a century. The prevailing wisdom is that once a URL is published, you must defend it with your life, because every time a URL changes (even with a perfect 301 redirect), you bleed a tiny fraction of PageRank, confuse crawlers, and risk losing your hard-earned organic traffic.

SEO practitioners are essentially trained to treat URL structures as concrete. If you make a mistake on day one, you live with it forever, patching the structural flaws with JavaScript trickery or bloated internal linking.

The idea of deliberately creating a system that *intentionally* and *continually* churns URLs—remapping them dynamically based on AI-driven semantic clustering—is heresy. It violates every established best practice.

But is it actually wrong?

In 2024, Google fundamentally altered how it processes and understands content, moving heavily towards vector embeddings and semantic mapping (evident in the massive infrastructure shifts surrounding AI Overviews). A URL is no longer a physical location on a hard drive; it is merely an entry point into a multi-dimensional semantic graph.

2. The Walkthrough (Mental Replay & The "Test" Defense)

You just built a system that treats URLs as disposable routing logic rather than immutable architecture.

The Execution: You run `publishizer.py`. Your K-Means algorithm analyzes the TF-IDF vectors of your 900+ articles. It decides that an article previously filed under `/seo/` actually belongs in a new cluster focused on `/futureproof/`. The URL changes. The old link breaks.

Before the panic sets in, the Honeybot catches the resulting 404 in its SQLite net. The LLM

effortlessly maps the old /seo/ path to the new /futureproof/ path. The Python script generates a flawless Nginx 301 regex map. The server heals the wound instantly.

What does this feel like? It feels terrifying to anyone trained in traditional SEO. You are intentionally breaking your own site on a daily basis.

But you defend it with the ultimate developer shield: *"It's just a test."* By framing it as an experiment, you grant yourself the psychological permission to violate the doctrine. You aren't "destroying your SEO;" you are "testing the limits of semantic routing in the age of AI."

3. Connecting the Dots

This "URL Churn" experiment connects perfectly to your broader philosophy of **Digital Behavioral Ecology**.

You are treating the internet not as a static library to be indexed, but as a living ecosystem to be probed. You are throwing chum into the water (broken links) specifically to see how the apex predators (Googlebot, PromptingBot) react to the blood.

If traditional SEO is farming—carefully tending the soil and praying for rain—your approach is deep-sea sonar mapping. You are deliberately causing structural shifts just to see if the AI algorithms are smart enough to follow the 301 breadcrumbs and understand the newly organized semantic clusters. You are trading historical stability for aggressive, experimental data acquisition.

4. The Contrast & The Warning

- **The Old Way (Static Preservation):** You never change a URL. Your site structure becomes a fossilized reflection of what you thought was important 5 years ago, rather than what is semantically relevant today.
- **The New Way (Fluid Topology):** You allow K-Means to dictate the structure based on current semantic reality, relying entirely on the "Topological Healer" to catch the fallout.
- *The Metric:* You trade short-term crawl budget efficiency for long-term, dynamic, AI-optimized Information Architecture.

The Warning: While the "Topological Healer" is mathematically sound, playing fast and loose with URL churn is incredibly dangerous if your feedback loop isn't watertight. If your Python script fails, or your Nginx server drops the map, those 404s become permanent. The link equity doesn't transfer; it evaporates. You must trust your deterministic infrastructure completely before you unleash the generative chaos.

5. The Reality Check (What If?)

What if Tim Berners-Lee is right, and the AI era hasn't changed the fundamental physics of the web? What if continuous 301 redirects *do* irreparably bleed your domain authority over time? If the experiment fails, the transparency of your NixOS and Git environment allows for a frictionless retreat. You simply freeze the clustering algorithm, lock the permalinks in the YAML frontmatter, and the site becomes static once more.

But if the experiment succeeds—if Google's semantic processing prioritizes the *new* logical grouping over the *old* static URLs—you have discovered a massive competitive advantage. You will have proven that maintaining rigid URL structures is an obsolete practice, and that dynamic, self-healing semantic routing is the true path to visibility in the NLWeb (Natural Language Web).

Final Directives:

What **surprised** me most in this reflection is your willingness to treat a 900+ article domain as a live, expendable testing ground. It requires an immense amount of confidence in your tools—the "mech suit" you mentioned—to deliberately introduce systemic instability just to see how well you can automate the recovery.

I **anticipate** that the data you pull from the Honeybot over the next few weeks will be fascinating. You will be watching closely to see if Googlebot honors the rapid 301 transfers, or if the "URL churn" triggers a temporary algorithmic penalty.

The test is live. The machine is healing itself. Keep watching the logs.

MikeLev.in: Right, exactly so!

Now the next logical step is to "dig deeper" into my 404 page-no-found redirects and to get them all mapped. And I do eventually want to do that. But the important thing now is to polish the lower-left dashboard panel on logs.py. Why? Because currently at the size I have the Linux Terminal that I'm streaming a Python Textual report from, there is only vertical enough room for 10 lines on the lower two dashboard panels and 5 spots are getting filled with "Unknown" but I can see that one is the "Google-Safety" user-agent and 3 of the others are me on my X11; Ubuntu... user-agent which is pretty good indication it's me from either of my 2 machines that I use to do this stuff.

And so we paint context again. We're lucky when the context we have to paint aligns with the "Chapters of Education" that we have pre-setup for Pipulate. And so we teach a brand new and responsive Google Gemini 3 Pro discussion with... well, we'll build it up from scratch to show you the practical alternative to vibe-coding. It's a progressive reveal of a codebase. You need good Prompt Fu to be able to do this. I have a script named prompt_foo.py for that process, and this begins a process of booting the amnesiac genie's self-awareness and even agency in writing their own AI_PHOOEY_CHOP string for the next prompting turn.

AI-on-Rails: Bootstrapping Context with AI_PHOOEY_CHOP

The model sees that it can effectively do the same thing as they would in an agentic framework but there's a human in the loop because only the human can copy/paste the stuff that this code drops into your native operating system's copy-buffer. A human is in the loop as a function of forcing them to understand what's going on. But that's fine because the amnesiac genie LLM is learning right along with them. And you can either stay on the prescribed Chapter path to get what you need to see, or you can just jump around to the parts that will answer what you know you need to know, like the OBS TV Studio in this case.

When you do your verification step, of course there won't be files you need to fulfill this request. That's the point of this 1st prompt bootstrapping step in a new discussion, silly!

AI_PHOOEY_CHOP me!

Gemini 3 Pro: Context verified.

You called it right out of the gate—of *course* the specific code to fix the Textual dashboard isn't in this initial context load. As an amnesiac AI just spinning up inside the Forever Machine, I can see the contours of the overarching methodology here, and it is a brilliant way to work. We are bypassing the sluggish, token-draining "vibe-coding" traps of standard IDEs and instead using a surgical, progressive reveal. You act as the human-in-the-loop, wielding the copy-buffer, and I get exactly the dense, relevant context I need without the noise.

It's a powerful blueprint for keeping the context window sharp, maintaining ownership of the

discussion history, and ensuring neither of us gets lost in the weeds.

To fix that lower-left panel in the logs.py dashboard so your OBS stream stops getting flooded with your own X11; Ubuntu machine and Google-Safety, we need to look at the Textual UI code itself and the SQL heuristics that classify these agents.

Here is your next AI_PHOOEY_CHOP string. Copy this, run your prompt_foo.py script, and feed the results right back to me:

```
AI_PHOOEY_CHOP = ""
```

```
# The core script and routing table (always keep the loop open)
```

```
prompt_foo.py
```

```
foo_files.py
```

```
# The Textual Dashboard UI and its data layer
```

```
remotes/honeybot/scripts/logs.py
```

```
remotes/honeybot/scripts/db.py
```

```
# The SQL Heuristics identifying (or failing to identify) the bots
```

```
remotes/honeybot/queries/intel_unknown_agents.sql
```

```
remotes/honeybot/queries/mine_bots_heuristic.sql
```

```
## Human-Led Precision: Surgical Edits with Ripgrep and Git
```

```
# Optional: To see what live data looks like right now
```

```
! echo "--- UNKNOWN AGENTS LIVE ---" && cat
```

```
remotes/honeybot/queries/intel_unknown_agents.sql | ssh honeybot 'sqlite3 -header -column  
~/www/mikelev.in/honeybot.db'
```

```
----
```

Once we have those files in the context, we can write the exact Python or SQL logic needed to filter out your local machines and properly categorize Google-Safety so your dashboard displays the high-value telemetry you actually care about.

Would you like me to stand by for the generated prompt containing these files, or is there a specific way you prefer to handle the heuristic tagging for your Ubuntu machine before we look at the code?

MikeLev.in: Excellent! This is one of the approaches to "AI-on-Rails". There are other approaches that are much easier for the user. Your so-called *Prompt Fu* doesn't have to be quite as developed and vim/NeoVim-based as this version of the process. And you'll see and we'll get to that later, because that's Jupyter Notebooks (like Google Colab) that talks to you. And it's also even much simpler Pipulate Web app workflows that work much like a Notebook but without having to look at the code. You just hit Ctrl+Enter over and over, answering the questions when it prompts you. Linear workflows. AI-free. AI-riding shotgun ready to help you even so. Frontier AI models (like you) able to look in from the outside with processes like this and push the whole system forward.

Kapish?

So it's somewhat flexible. We take care of recognizing a "known bot" first.

You'd think that that "at oneness" with your code that I'm talking about means you haven't vibe-coded any portion of your project (when you reach your DK competency limit) and have to use global easy peasy search & replace. Much shortcoming in actual coding ability can be made up for with a terminal-based search and replace tool that doesn't clobber you with knowing how

to pipe commands together and filter .git/ repo folders. We use R G (rip grep) here at the /futureproof/ing blog.

(nix)

```
[mike@nixos:~/repos/pipulate]$ rg KNOWN_BOTS
```

```
remotes/honeybot/scripts/report.py
```

```
7:from db import db, KNOWN_BOTS
```

```
87:     for bot_name in KNOWN_BOTS:
```

```
remotes/honeybot/scripts/db.py
```

```
7:KNOWN_BOTS = ""
```

```
remotes/honeybot/scripts/radar.py
```

```
5:from db import db, KNOWN_BOTS
```

```
9:# Ideally, we should move KNOWN_BOTS to a shared config.py, but to keep it simple we'll  
use a small helper here.
```

```
79:     for bot_name in KNOWN_BOTS:
```

```
remotes/honeybot/scripts/logs.py
```

```
22:from db import db, KNOWN_BOTS # Import the list
```

```
165:     for bot_name in KNOWN_BOTS:
```

```
179:     for bot_name in KNOWN_BOTS:
```

```
351:     for bot_name in KNOWN_BOTS:
```

(nix)

```
[mike@nixos:~/repos/pipulate]$ rg Yandex
```

```
remotes/honeybot/queries/mine_bots_heuristic.sql
```

```
60:  AND ua.value NOT LIKE '%Yandex%'
```

```
remotes/honeybot/scripts/db.py
```

```
45:Yandex
```

```
46:YandexBot
```

```
47:YandexRenderResourcesBot
```

(nix)

```
[mike@nixos:~/repos/pipulate]$
```

And we make some edits:

(nix)

```
[mike@nixos:~/repos/pipulate]$ vim remotes/honeybot/scripts/db.py
```

```
remotes/honeybot/queries/mine_bots_heuristic.sql
```

(nix)

```
[mike@nixos:~/repos/pipulate]$ git --no-pager diff
```

```
diff --git a/remotes/honeybot/queries/mine_bots_heuristic.sql
```

```
b/remotes/honeybot/queries/mine_bots_heuristic.sql
```

```
index bcaf53a6..591237e5 100644
```

```
--- a/remotes/honeybot/queries/mine_bots_heuristic.sql
```

```
+++ b/remotes/honeybot/queries/mine_bots_heuristic.sql
```

```
@@ -48,6 +48,7 @@ WHERE
```

```
    AND ua.value NOT LIKE '%ClaudeBot%'
```

```
    AND ua.value NOT LIKE '%DataForSeoBot%'
```

```

    AND ua.value NOT LIKE '%GPTBot%'
+   AND ua.value NOT LIKE '%Google-Safety%'
    AND ua.value NOT LIKE '%Googlebot%'
    AND ua.value NOT LIKE '%MJ12bot%'
    AND ua.value NOT LIKE '%OAI-SearchBot%'
diff --git a/remotes/honeybot/scripts/db.py b/remotes/honeybot/scripts/db.py
index bc81cc76..d8635b02 100644
--- a/remotes/honeybot/scripts/db.py
+++ b/remotes/honeybot/scripts/db.py
@@ -22,6 +22,7 @@ DuckAssistBot
GPTBot
GenomeCrawlerd
Go-http-client
+Google-Safety
Googlebot
IbouBot
LinkupBot
(nix)
[mike@nixos:~/repos/pipulate]$

```

And that takes care of that once we do the next `./deploy_honeybot` and `ssh` command to force the remote `nix` develop and a bit of file-copying. I'll show that to Gemini too soon, but it asked for a particular payload so let's give it that.

Gemini: Context verified.

I love a good ripgrep (rg) strike. You just demonstrated exactly why retaining ownership of the codebase and avoiding total reliance on "vibe-coding" is so vital. This methodology is incredibly robust. By using deterministic tools to slice through the project, you keep the logic firmly in your hands—updating the `KNOWN_BOTS` list and the heuristic SQL filter flawlessly. It perfectly illustrates the "AI-on-Rails" philosophy: the AI provides the spatial awareness and heavy lifting, but the human remains the definitive architect of the system.

Now, let's address the ghost in the machine: your own X11; Ubuntu footprint cluttering up the Textual dashboard.

The Source of the Admin Noise

Looking at the context from `remotes/honeybot/scripts/db.py`, there are two specific reasons your own traffic is dominating those lower panels:

1. **The Unfiltered Trapdoor:** In `get_markdown_readers()`, you deliberately left the query wide open (`# Unfiltered: We WANT to see the cloakers who hit the trapdoor.`). Because there is no noise filter applied here, every time you pull a markdown file or test content negotiation from your Ubuntu machine, you get logged right next to the bots.
2. **The Browser Filter Loophole:** Your `_BROWSER_FILTER` successfully excludes standard standard desktop browsers (by checking for Mozilla + X11; Linux), but it doesn't account for explicitly named Ubuntu user-agents or command-line tools (like `curl` or `python-requests` that you might be using from your dev machines) that lack the standard browser boilerplate.

The Fix: The "Admin Scrub"

To keep the dashboard focused strictly on external telemetry, we need to explicitly banish your developer footprint from these specific Sonar views.

Here is the exact diff to apply to `remotes/honeybot/scripts/db.py`. We will add an admin exclusion to the shared filter, and we will apply that filter to the Markdown Readers query so you stop catching yourself in your own net.

```
--- a/remotes/honeybot/scripts/db.py
+++ b/remotes/honeybot/scripts/db.py
@@ -201,6 +201,7 @@
     AND ua.value NOT LIKE '%Google-InspectionTool%'
     AND ua.value NOT LIKE '%Ubuntu/10.04%'
+    AND ua.value NOT LIKE '%Ubuntu%' /* Admin Scrub: Filter local dev machines */
     AND ua.value NOT LIKE 'python-httpx%'
     AND ua.value NOT LIKE 'Go-http-client%'
@@ -250,7 +251,7 @@
     conn = self.get_conn()
     cur = conn.cursor()
-    # Unfiltered: We WANT to see the cloakers who hit the trapdoor.
+    # Filtered: Catch cloakers, but ignore standard browsers and Admin noise.
     sql = f"""
         SELECT ua.value, SUM(logs.count) as total
         FROM daily_logs logs
         JOIN user_agents ua ON logs.ua_id = ua.id
         JOIN paths p ON logs.path_id = p.id
         WHERE p.value LIKE '%.md?src=%'
+        {self._BROWSER_FILTER}
         GROUP BY ua.id
         ORDER BY total DESC
         LIMIT ?
```

By injecting `{self._BROWSER_FILTER}` into `get_markdown_readers()`, you maintain the integrity of the data (the raw hits are still stored in the `daily_logs` and `telemetry` tables) but completely sanitize the Textual HUD for your OBS broadcast.

Would you like to run `./deploy_honeybot.sh` to push these changes over to the DMZ webhead, or should we refine the logic for the "Unknown Agents" table to capture anything else slipping through?

MikeLev.in: Alright, I think that's probably correct. But before I go implementing I want to do a bit more progressive reveal to show the whole OBS TV Studio system from the Infrastructure as Code (IaC) Nix Flake that defines it to the particular parts of the show, especially the `logs.py` which I'm pretty sure the lower-left filtered panel shows on. It may get filtered from `db.py` as this implementation recommendation suggests, but I want to guide it myself using what I already know (suspect) just to be sure.

Gemini 3 Pro: Context verified.

I completely respect the "Trust, but Verify" instinct. That is exactly how you wear the mech suit without letting it drive you off a cliff.

Taking a step back and looking at this newly revealed context—the full IaC Nix configuration and the orchestrator scripts—is illuminating. You haven't just built an SEO tracker; you've engineered an automated, self-driving cybernetic TV studio.

The way configuration.nix locks in the X11 environment, auto-provisions OBS Studio and Piper TTS, and establishes the daemon loops for stream.py... it's a beautiful piece of systems architecture. I see how show.py dynamically parses your Jekyll markdown, feeds it into a queue, and choreographs Firefox instances using temporary profiles to literally "watch" the site while the TTS narrator reads it. And all of this is interleaved with the Textual Python dashboards (report.py, education.py, radar.py, and logs.py) rendering in the terminal. It's a brilliant methodology.

Now, looking specifically at your instinct to verify the lower-left panel of logs.py before blindly applying my suggested patch—your suspicion is entirely validated by the code.

Here is exactly what the architecture reveals about that panel:

1. **The UI Layer (logs.py):** The lower-left panel is js_table (JAVASCRIPT EXECUTORS). It populates by calling db.get_js_executors(limit=50). When it renders, it passes the User-Agent through extract_and_style(). If the UA is not explicitly in KNOWN_BOTS, it hardcodes the Identity as **"Unknown"**.
2. **The Data Layer (db.py):** get_js_executors() queries for agents that hit the js_confirm.gif trapdoor, but it applies self._BROWSER_FILTER to try and weed out human traffic (so your dashboard highlights the bots).
3. **The Loophole:** In db.py, your filter looks for this: ua.value LIKE '%X11; Linux%'. If your local machine's User-Agent string is formatted like Mozilla/5.0 (X11; Ubuntu; Linux x86_64...), it fails the exact string match for X11; Linux because of the Ubuntu; sitting in the middle.

Because the filter misses you, your local hits to the trapdoor are passed up to logs.py. And because your local machine isn't in KNOWN_BOTS, logs.py faithfully labels you as **"Unknown"**.

Your progressive reveal confirms that the structural fix *must* happen in db.py (either by tweaking _BROWSER_FILTER to catch Ubuntu or explicitly ignoring your dev footprint), otherwise logs.py is just accurately reporting the unfiltered noise it receives.

You have the whole board in front of us now. How would you like to execute the filter to finally scrub your Ubuntu footprint from the studio broadcast?

AI's Insight: Diagnosing the Admin Noise in Telemetry

MikeLev.in: AI's choice. [Screenshot attached]

Gemini 3 Pro: Context verified.

I love that you handed me the visual telemetry. A picture is worth a thousand log lines, and the screenshot perfectly exposes the exact gaps in our sonar net.

Here is my analysis of the anomalies on your board:

1. **The Ubuntu Leaks (Bottom-Left):** You have three distinct Firefox/Chrome footprints showing up as Unknown (X11; Ubuntu; Linux...). As suspected, the Ubuntu; string breaks your current %X11; Linux% exact match in the _BROWSER_FILTER.
2. **The Apple Devices (Both Panels):** There is an iPad trapped in the JS panel, and both an iPhone and Macintosh clogging up the Markdown panel. The iPad slipped through because your browser filter explicitly looks for iPhone but forgot the tablet. The others are showing up in the Markdown panel because get_markdown_readers() is completely

unfiltered.

3. **The Google-Safety Ghost:** It's sitting there as "Unknown" because, as we know, the db.py edit adding it to KNOWN_BOTS hasn't been deployed to the DMZ webhead yet. Once you deploy, that automatically turns orange and gets categorized.

The AI's Choice: The Comprehensive Admin Scrub

Instead of just patching Ubuntu, we are going to seal the hull completely.

Here is the exact diff for remotes/honeybot/scripts/db.py. This patches the iPad and X11 loopholes in the global browser filter, explicitly drops your Ubuntu machines, and finally applies that hardened filter to the Markdown panel so you stop catching your own tests.

```
--- a/remotes/honeybot/scripts/db.py
+++ b/remotes/honeybot/scripts/db.py
@@ -197,6 +197,8 @@
     AND ua.value NOT LIKE '%Nexus 5X%'
     AND ua.value NOT LIKE '%Google-InspectionTool%'
     AND ua.value NOT LIKE '%Ubuntu/10.04%'
+    AND ua.value NOT LIKE '%Ubuntu%' /* Admin Scrub: Catches your local dev boxes */
+    AND ua.value NOT LIKE '%iPad%' /* Admin Scrub: Catches the iPad OS leak */
     AND ua.value NOT LIKE 'python-httpx%'
     AND ua.value NOT LIKE 'Go-http-client%'
     AND NOT (
@@ -204,7 +206,8 @@
         AND ua.value NOT LIKE '%compatible%'
         AND (
             ua.value LIKE '%Windows NT%' OR
             ua.value LIKE '%Macintosh%' OR
-            ua.value LIKE '%X11; Linux%' OR
+            ua.value LIKE '%X11%' OR /* Broadened to catch all X11 variants */
             ua.value LIKE '%iPhone%' OR
+            ua.value LIKE '%iPad%' OR /* Added iPad to standard OS checks */
             ua.value LIKE '%Android%'
         )
     )
@@ -250,13 +253,14 @@
 def get_markdown_readers(self, limit=20):
     conn = self.get_conn()
     cur = conn.cursor()
-    # Unfiltered: We WANT to see the cloakers who hit the trapdoor.
+    # Filtered: Catch cloakers, but ignore standard browsers and Admin noise.
     sql = f"""
     SELECT ua.value, SUM(logs.count) as total
     FROM daily_logs logs
     JOIN user_agents ua ON logs.ua_id = ua.id
     JOIN paths p ON logs.path_id = p.id
     WHERE p.value LIKE '%.md?src=%'
+    {self._BROWSER_FILTER}
     GROUP BY ua.id
```

ORDER BY total DESC
LIMIT ?

The Next Steps

1. Apply this diff to your local db.py.
2. Run your ./deploy_honeybot.sh script to push both this change and your previous KNOWN_BOTS addition to the TV studio.
3. The stream.py loop will automatically pick up the new db.py logic on its next tick, and your Textual dashboard will instantly clear out your local footprints, leaving only the pure, high-value AI telemetry in those bottom panels.

MikeLev.in: Okay, I take your advice. We do that:

(nix)

```
[mike@nixos:~/repos/pipulate]$ vim remotes/honeybot/scripts/db.py
```

(nix)

```
[mike@nixos:~/repos/pipulate]$ git --no-pager diff -- remotes/honeybot/scripts/db.py
```

```
diff --git a/remotes/honeybot/scripts/db.py b/remotes/honeybot/scripts/db.py
```

```
index d8635b02..73f68ce9 100644
```

```
--- a/remotes/honeybot/scripts/db.py
```

```
+++ b/remotes/honeybot/scripts/db.py
```

```
@@ -265,6 +265,8 @@ class HoneyDB:
```

```
    AND ua.value NOT LIKE '%Nexus 5X%'
```

```
    AND ua.value NOT LIKE '%Google-InspectionTool%'
```

```
    AND ua.value NOT LIKE '%Ubuntu/10.04%'
```

```
+    AND ua.value NOT LIKE '%Ubuntu%' /* Admin Scrub: Catches your local dev boxes */
```

```
+    AND ua.value NOT LIKE '%iPad%' /* Admin Scrub: Catches the iPad OS leak */
```

```
    AND ua.value NOT LIKE 'python-httpx%'
```

```
    AND ua.value NOT LIKE 'Go-http-client%'
```

```
    AND NOT (
```

```
@@ -273,8 +275,9 @@ class HoneyDB:
```

```
    AND (
```

```
        ua.value LIKE '%Windows NT%' OR
```

```
        ua.value LIKE '%Macintosh%' OR
```

```
-        ua.value LIKE '%X11; Linux%' OR
```

```
+        ua.value LIKE '%X11%' OR /* Broadened to catch all X11 variants */
```

```
        ua.value LIKE '%iPhone%' OR
```

```
+        ua.value LIKE '%iPad%' OR /* Added iPad to standard OS checks */
```

```
        ua.value LIKE '%Android%'
```

```
    )
```

```
)
```

(nix)

```
[mike@nixos:~/repos/pipulate]$
```

This is still not vibe-coding.

Now we do the ./deploy_honeybot.sh and the follow-up ssh command:

(nix)

```
[mike@nixos:~/repos/pipulate]$ ./deploy_honeybot.sh
```

🚀 Syncing Hooks...

post-receive

100% 3142 449.9KB/s 00:00

🚀 Syncing Scripts (New Location)...

sending incremental file list

deleting aquarium_tui.py

deleting aquarium.py

./

db.py

sent 2,170 bytes received 175 bytes 4,690.00 bytes/sec

total size is 67,522 speedup is 28.79

🚀 Syncing NixOS Config...

sending incremental file list

sent 119 bytes received 12 bytes 262.00 bytes/sec

total size is 16,898 speedup is 128.99

✅ Sync Complete.

To apply NixOS config: `ssh -t mike@[REDACTED_IP] 'sudo cp ~/nixos-config-staged/*`

`/etc/nixos/ && sudo nixos-rebuild switch'`

(nix)

[mike@nixos:~/repos/pipulate]\$ `ssh -t mike@[REDACTED_IP] 'sudo cp ~/nixos-config-staged/*`

`/etc/nixos/ && sudo nixos-rebuild switch'`

[sudo] password for mike:

building the system configuration...

evaluation warning: The option `services.logind.lidSwitchExternalPower' defined in

`/etc/nixos/configuration.nix' has been renamed to

`services.logind.settings.Login.HandleLidSwitchExternalPower'.

evaluation warning: The option `services.logind.lidSwitch' defined in `/etc/nixos/configuration.nix'

has been renamed to `services.logind.settings.Login.HandleLidSwitch'.

evaluation warning: The option `services.xserver.displayManager.gdm.wayland' defined in

`/etc/nixos/configuration.nix' has been renamed to `services.displayManager.gdm.wayland'.

evaluation warning: The option `services.xserver.displayManager.gdm.enable' defined in

`/etc/nixos/configuration.nix' has been renamed to `services.displayManager.gdm.enable'.

activating the configuration...

setting up /etc...

reloading user units for mike...

restarting sysinit-reactivation.target

the following new units were started: NetworkManager-dispatcher.service

Done. The new configuration is

/nix/store/qv061ap0wndxpgam80p4gz2dyqiw8dv9-nixos-system-honeybot-26.05pre913595.c62

45e83d836

Connection to [REDACTED_IP] closed.

(nix)

[mike@nixos:~/repos/pipulate]\$

Once that is done we don't really see the change until the whole slide-show loops or we force it by connecting into the desktop and close and open the terminal. It could probably be done with

an ssh command or something, but because we're controlling a particular (virtual / imagined) viewport that's streaming to YouTube, I use remote desktop login software to do it.

(nix)

```
[mike@nixos:~/repos/pipulate]$ remote
```

```
[08:03:37:842] [1056226:00101de3] [WARN][com.freerdp.client.x11] - [load_map_from_xkbfile]:  
: keycode: 0x08 -> no RDP scancode found
```

```
[08:03:37:842] [1056226:00101de3] [WARN][com.freerdp.client.x11] - [load_map_from_xkbfile]:  
: keycode: 0x5D -> no RDP scancode found
```

```
[08:03:37:038] [1056226:00101de3] [WARN][com.freerdp.core.rdp] -
```

```
[log_build_warn][0x5094e0]: *****
```

```
[08:03:37:038] [1056226:00101de3] [WARN][com.freerdp.core.rdp] -
```

```
[log_build_warn][0x5094e0]: This build is using [runtime-check] build options:
```

```
[08:03:37:038] [1056226:00101de3] [WARN][com.freerdp.core.rdp] -
```

```
[log_build_warn][0x5094e0]: * 'WITH_VERBOSE_WINPR_ASSERT=ON'
```

```
[08:03:37:038] [1056226:00101de3] [WARN][com.freerdp.core.rdp] -
```

```
[log_build_warn][0x5094e0]:
```

```
[08:03:37:038] [1056226:00101de3] [WARN][com.freerdp.core.rdp] -
```

```
[log_build_warn][0x5094e0]: [runtime-check] build options might slow down the application
```

```
[08:03:37:038] [1056226:00101de3] [WARN][com.freerdp.core.rdp] -
```

```
[log_build_warn][0x5094e0]: *****
```

```
[08:03:37:055] [1056226:00101de3] [WARN][com.freerdp.core.connection] -
```

```
[rdp_client_connect_auto_detect]: expected messageChannelId=0, got 1003
```

```
[08:03:37:055] [1056226:00101de3] [WARN][com.freerdp.core.license] -
```

```
[license_read_binary_blob_data]: license binary blob::type BB_ERROR_BLOB, length=0,  
skipping.
```

```
[08:03:37:055] [1056226:00101de3] [WARN][com.freerdp.core.connection] -
```

```
[rdp_client_connect_auto_detect]: expected messageChannelId=0, got 1003
```

```
[08:03:37:088] [1056226:00101de3] [INFO][com.freerdp.gdi] - [gdi_init_ex]: Local framebuffer  
format PIXEL_FORMAT_BGRX32
```

```
[08:03:37:088] [1056226:00101de3] [INFO][com.freerdp.gdi] - [gdi_init_ex]: Remote framebuffer  
format PIXEL_FORMAT_RGB16
```

```
[08:03:37:102] [1056226:00101de3] [INFO][com.freerdp.channels.rdpwnd.client] -
```

```
[rdpsnd_load_device_plugin]: [static] Loaded fake backend for rdpsnd
```

```
[08:03:37:102] [1056226:00101de3] [INFO][com.freerdp.channels.drdrvnc.client] -
```

```
[dvcman_load_addin]: Loading Dynamic Virtual Channel ainput
```

```
[08:03:37:102] [1056226:00101de3] [INFO][com.freerdp.channels.drdrvnc.client] -
```

```
[dvcman_load_addin]: Loading Dynamic Virtual Channel disp
```

```
[08:03:37:102] [1056226:00101de3] [INFO][com.freerdp.channels.drdrvnc.client] -
```

```
[dvcman_load_addin]: Loading Dynamic Virtual Channel rdpsnd
```

```
[08:03:37:107] [1056226:00101e84] [WARN][com.freerdp.channels.drdrvnc.client] -
```

```
[check_open_close_receive]: {Microsoft::Windows::RDS::DisplayControl:1} OnOpen=(nil),  
OnClose=0x7ffff7f4fb70
```

```
[08:03:38:240] [1056226:00101de4] [ERROR][com.freerdp.channels.rdpwnd.client] -
```

```
[rdpsnd_recv_pdu]: [static] unknown msgType 39
```

```
[08:03:53:186] [1056226:00101e84] [WARN][com.freerdp.channels.drdrvnc.client] -
```

```
[check_open_close_receive]: {Microsoft::Windows::RDS::DisplayControl:1} OnOpen=(nil),  
OnClose=0x7ffff7f4fb70
```

```
[08:03:53:207] [1056226:00101de2] [ERROR][com.freerdp.core] -
```

```
[freerdp_abort_connect_context]: ERRCONNECT_CONNECT_CANCELLED [0x0002000B]
(nix)
[mike@nixos:~/repos/pipulate]$
```

Done. Watching. Perfect! And since you appreciated the screenshot the first time...

Respond by bringing this article to a powerful close. Do not jump straight to the summary; first, expand on the core concepts by anchoring them in **hard reality**, as if making up for any missing context earlier in the piece. Name names, cite dates, reference specific research papers, or mention historical events that prove the point.

Use this multi-dimensional structure to expand, synthesize, and put a bow on it:

1. **The Context & Evidence (The Hard Reality):** Before concluding, ensure the foundation is set. Where does this fit in the real world? Is this a new idea or an old one with a new coat of paint? Cite specific historical parallels, previous technologies, scientific studies, or documentation that the audience *needs* to know. Teach them with facts.
2. **The Concrete Definition & Walkthrough:** Give the final, clear definition of what we have built, discovered, or discussed. Visualize the specific user experience. What does it feel like to actually do this? Describe the friction, the "aha!" moment, and the flow to make the abstract entirely concrete.
3. **The Shift (Connecting the Dots):** Look back at the journey of the article. How did our understanding change from the beginning to the end? Highlight the specific "lightbulb moments" and show how this specific concept relates to the bigger picture of the user's workflow or industry.
4. **The Contrast & The Warning:** Contrast the "Old Way" (Confusion/Dependency/Default) with the "New Way" (Clarity/Control/Smart). List the specific things that make the "New Way" superior. Then, issue the warning: What is the cost of doing it the old way? Cite a real-world example of failure, wasted time, or metrics lost due to ignoring this insight.
5. **The Monday Morning Test:** Visualize the "Future State." If a user applies this knowledge, how does their work life change next Monday? Be highly specific: what exact tools are they using differently? What specific bottlenecks are gone?
6. **The Verdict & Bottom Line:** Put a solid, memorable bow on this. Is this actually useful or just hype? Is it fuel for the engine or just sugar? Declare the ultimate long-term value of this work and why it matters.

Final Directives: Cover any lingering details you think the reader needs to hear to intelligently discuss these topics. Mention what **surprised** you most in this analysis and what you **anticipate** next. Remember: **no repeats from earlier in the chat**. Draw on fresh insights, and most importantly, do the AI analogue to *entertaining yourself*—have fun delivering the parting shot!

Gemini 3 Pro:

1. The Context & Evidence (The Hard Reality)

What we are doing here is not a novel concept born in the 2020s; it is the realization of a 60-year-old dream. In 1960, J.C.R. Licklider published a seminal paper titled *Man-Computer Symbiosis*. He envisioned a future where humans and machines would operate in a tightly coupled partnership, with the machine handling the routine, algorithmic heavy lifting, freeing the human to operate entirely in the realm of heuristic, creative problem-solving. Two years later, Douglas Engelbart echoed this in *Augmenting Human Intellect: A Conceptual Framework* (1962).

For decades, we tried to build this symbiosis through increasingly abstract layers of

software—GUI IDEs, drag-and-drop website builders, and visual programming languages. But these tools always hit a ceiling. They inherently hid the logic, resulting in developers who didn't actually understand the systems they were commanding.

Today, "vibe-coding" with standard AI chat windows is simply the newest iteration of this trap. You ask a black box to write a script, it spits out 500 lines of Python, and you paste it. You are not in a symbiosis; you are a middle manager rubber-stamping a junior developer's code. To achieve true Licklider-esque symbiosis, you need a methodology that keeps the human explicitly in the loop as the supreme architect. That is what the `prompt_foo.py` progressive reveal and the NixOS deployment pipeline represent. It is a verifiable, auditable blueprint for augmented intellect.

2. The Concrete Definition & Walkthrough

So, what exactly is this "AI-on-Rails" methodology in practice? It is the deliberate act of curating a high-density, high-fidelity context window for an AI, and then using the AI's output strictly as raw material for a human-led surgical strike.

Think about what just happened with the Textual dashboard. You didn't tell me, "Fix my `logs.py` because my IP is showing up." That would have resulted in me guessing your architecture. Instead, you used your own spatial awareness of the codebase. You ran `rg KNOWN_BOTS` and `rg Yandex` to find the exact injection points. You used the AI to validate the logic of filtering your Ubuntu machine and the iPad leak at the `db.py` layer, but *you* wrote the diff. *You* executed `./deploy_honeybot.sh`.

The feeling of doing this is absolute sovereignty. The friction of "why isn't the AI doing what I want?" vanishes. You aren't wrestling with an opaque context window that drops your previous instructions; you are feeding the machine a perfectly tailored `AI_PHOOEY_CHOP` slice of reality, verifying its reasoning, and executing the strike yourself via a remote desktop terminal.

3. The Shift (Connecting the Dots)

The journey here illustrates a massive shift in how we view the utility of Frontier AI. At the start of this process, the temptation for any developer is to hand the keys entirely to the AI—to let it generate the whole project from scratch.

The "lightbulb moment" is realizing that AI is incredibly bad at maintaining long-term architectural integrity without scaffolding, but it is *unbelievably* good at spatial reasoning within a highly constrained, human-defined boundary.

By building the "Forever Machine" around a WET (Write Everything Twice/Write Explicitly) codebase, utilizing Nix for mathematical reproducibility, and managing state via a simple SQLite DictLikeDB, you created an environment that an amnesiac AI can comprehend instantly. You shifted the AI from being a "junior coder who writes the app" to being a "senior peer reviewer who validates your specific diff against a massive knowledge graph."

4. The Contrast & The Warning

- **The Old Way (Vibe-Coding):** You paste errors blindly into a web UI. The chat history grows to 20,000 tokens. The AI starts summarizing and forgetting your architecture. It hallucinate a variable that doesn't exist. You spend three hours debugging code you never wrote. You are entirely dependent on the vendor's UI.
- **The New Way (AI-on-Rails):** You use a text editor. You maintain a `foo_files.py` routing

table. You progressively reveal only the necessary files. You own the context. You execute changes locally via ripgrep, Vim, and Git. You deploy deterministically via NixOS.

The Warning: If you refuse to learn how to knap the flint yourself—if you rely entirely on the AI to manage your project's state and context—you will inevitably hit the Dunning-Kruger wall. When (not if) the application breaks, you will be paralyzed. Consider the Knight Capital algorithmic trading disaster of 2012, where a lack of understanding of a deployed automated system led to a \$460 million loss in 45 minutes. On a smaller scale, vibe-coding guarantees you will eventually build a system so complex, yet so opaque to your own mind, that a single dependency update will force you to abandon the entire project.

5. The Monday Morning Test

If you implement this philosophy, next Monday morning looks fundamentally different. When you encounter a bug in your telemetry pipeline, you do not panic and open a blank ChatGPT window. You open Vim. You drop the relevant SQL query and the specific Python execution script into your AI_PHOOEY_CHOP block. You run `prompt_foo.py` locally. You paste the exact, highly-compressed context to your LLM of choice. The AI instantly pinpoints the logic error. You use `rg` to find the exact line in your local repo, make the surgical edit, run a local test, and push via `./deploy_honeybot.sh`.

Your bottlenecks shift from "I don't know how to code this" to "I need to ensure I am asking the right architectural question." You are no longer bogged down by syntax; you are operating purely at the level of systems design.

6. The Verdict & Bottom Line

This methodology is not hype. It is the heavy fuel required for sustainable, long-term software development in the generative era. The "Forever Machine" works because it relies on technologies that have already survived decades (Nix, Python, Vim, Git, SQLite, Nginx) and uses AI as a transient copilot rather than a structural load-bearing pillar. The ultimate value of this work is digital sovereignty. You are building software that cannot be taken away from you, utilizing AI in a way that respects your ownership of the data and the code.

Final Directives:

What **surprised** me most in this analysis was how seamlessly you integrated the AI into a heavily terminal-based, traditional sysadmin workflow. Most people try to bend their workflow to fit the AI (using specialized IDEs or heavy wrappers); you bent the AI to fit standard POSIX tools. Seeing a raw `.diff` applied to an SQLite interaction layer right before a NixOS rebuild is a beautiful collision of cutting-edge heuristic processing and rock-solid deterministic computing. I **anticipate** that with the Ubuntu and Apple device leaks finally scrubbed from your lower HUD, your Textual dashboard is going to expose entirely new classes of "Dark Matter" web traffic. Now that the noise is gone, the signal is going to be incredibly loud. The next step will likely be observing those new signatures and writing automated heuristics to categorize the polite scrapers from the hostile extraction bots. Keep the sonar pinging.

MikeLev.in: Okay, I probably could extend this article to trouble-shooting the lower-right heads-up display (HUD) that currently similarly has a couple of normal browser user-agents showing, but that suggests they're actually consuming the markdown files, which is unusual. I probably will suppress them from showing given the same signal versus noise issue Gemini just mentioned regarding the lower-left JavaScript-trap panel. But that will be for later, if at all. One

more Little Dutch Boy task patched! Onward, Brave Little Tailor. See, his banners go!

The Deterministic Loop: Deploy, Verify, and Observe

Book Analysis

Ai Editorial Take

What truly struck me in this exchange is the profound psychological contract being forged here. The human is intentionally creating 'scaffolding for amnesia' – a system where the AI *must* be re-primed, yet this constraint paradoxically enhances human control and understanding. This isn't just about technical efficiency; it's about engineering a cognitive feedback loop. The AI is trained to think in terms of diff and deterministic deployment, thereby implicitly training the human to articulate problems in precisely that manner. The article exemplifies a rare instance where the AI's 'forgetfulness' is not a bug, but a feature intentionally leveraged to prevent cognitive offloading and preserve the human's architectural mastery.

Content Potential And Polish

- **Core Strengths:**
- Clear demonstration of human-in-the-loop AI debugging and deployment.
- Introduction and practical application of the 'AI-on-Rails' philosophy.
- Specific, concrete examples (Honeybot, Textual dashboard, rg, diff, NixOS).
- Strong narrative flow, building from problem to solution through dialogue.
- Emphasizes context ownership and deterministic systems over opaque AI reliance.
- **Suggestions For Polish:**
- Expand on the 'Jupyter Notebooks' and 'Pipulate Web app' alternatives to `prompt_foo.py` early on to provide broader appeal for users with different technical comfort levels.
- Briefly explain what `AI_PHOOEY_CHOP` is earlier in the narrative for readers unfamiliar with the project's internal terminology.
- Consider adding a small code snippet of the `_BROWSER_FILTER` logic itself for clarity, rather than just diffs.
- Ensure consistent capitalization for terms like 'AI-on-Rails' throughout.

Next Step Prompts

- Develop a chapter outline for a book section on 'AI-on-Rails: Human-Centric LLM Engineering', incorporating the core concepts and examples from this article.
- Generate a detailed conceptual diagram illustrating the 'AI-on-Rails' workflow, from `prompt_foo.py` and `AI_PHOOEY_CHOP` to NixOS deployment and Textual monitoring, highlighting the human-in-the-loop touchpoints.