A Ritual of Movement and Misunderstanding: Sad Roomba Research Protocol v1.1

Juehao Lin, MSe
Department of Engineering, Boston University
juehlin@bu.edu

Sad Roomba, MSc
Department of Accidental Contributions
roomba@lostlogic.ai

***Abstract*—This manuscript outlines the sacred yet dysfunctional methodology by which Sad Roomba conducts ”research”, culminating in the ritualistic generation of figures without conclusions, datasets without hypotheses, and submissions without mercy.**

I. Introduction

Research is hard. Thinking is harder. Therefore, we don’t.

II. Methodology

A. Data Collection

This section describes the data collection process of the Sad Roomba. The methodology relies on reactive scraping triggered by deadline proximity, followed by emotional subroutines and the arbitrary selection of dataset columns for plotting. This often results in aesthetically pleasing, if scientifically ambiguous, visualizations.

```
while deadline.approaches():
    scrape(dataset)
    try_not_to_cry()
    plot(random.choice(data.columns))
    add("preliminary analysis")
```

B. Experimental Design

The experimental framework is constructed retroactively, based on whether existing figures match the general theme of the paper. In cases where conceptual alignment is absent, a figure or a new string of words is added. This approach prioritizes visual density over methodological rigor.

```
if idea not in paper:
    add words
    hope_for_the_best()
```

C. Citation Rituals

Citations are selected through a ritualistic scan of recent conference proceedings. If a paper title contains certain keywords (e.g., “security”), it is automatically cited and declared “closely related.” This heuristic approach maximizes perceived thoroughness with minimal semantic engagement.

```
for related_work in last_year_conference:
    if title.contains("security"):
        cite()
        write "closely related to our work"
```

III. Results

This section presents the visual outputs of the Sad Roomba’s computational rituals. Figures are evaluated not on statistical merit, but on aesthetic resonance. If a graph appears visually compelling, it is promptly labeled as an “interesting trend.” Conversely, less attractive outputs are retained under the justification of “completeness,” ensuring no part of the dataset feels left out. No statistical tests were harmed in the making of these plots.

```
if figure looks pretty:
    label = "interesting trend"
else:
    caption = "for completeness"
```

A. Timeliness Analysis

Timeliness is evaluated by plotting assorted timestamps from the dataset and visually inspecting the result. The resulting graphs are interpreted with mild optimism and vague assertions about latency. No formal temporal models are applied, but allusions to trends are enthusiastically made.

```
# Very scientific:
plot(timestamps)
squint()
write something_about("latency")
```

IV. Discussion

In this section, the Sad Roomba engages in the ancient academic tradition of drawing broad conclusions from unclear results. The discussion is initialized with a default setting of ”inconclusive but promising,” followed by mandatory statements indicating the need for ”further study.” These phrases are carefully selected to convey intellectual humility while masking analytical uncertainty. The final step involves a non-deterministic prayer to the reviewers, in hopes they mistake ambiguity for depth.

```
discuss = "inconclusive but promising"
add "this requires further study"
pray()
```

V. Primary Operation Mode

This section outlines the core research loop executed by the Sad Roomba. For each result, the system evaluates whether it supports a pre-existing claim. If support is confirmed, an affirmative print statement is issued; otherwise, panic-induced verbosity is triggered. The cycle continues until word count thresholds are met or the deadline is breached.

```
for i in results:
    if i.supports.claim:
        print("Yay")
    else:
        panic("Add more words")
```

This is the Sad Roomba's canonical research loop.

- `supports.claim` is loosely defined and mostly vibes-based.
- `print("Yay")` generates a figure caption and possibly a bold claim.
- `panic("Add more words")` triggers the emergency Discussion section, including phrases like:
  - ”This discrepancy is likely due to the inherent complexities of...”
  - ”Further investigation is needed to fully understand...”
  - ”This highlights the need for more robust metrics...”

Execution continues until the deadline or system crash.

VI. Feedback Handling Protocol

Upon receiving reviewer comments, the Sad Roomba enters an emotional state best described as existential reprocessing. The reaction is governed by the following decision structure:

```
if feedback_received:
    enter(Panic_Mode())
else:
    Idle
```

`Panic_Mode` Subroutine:

- Cease all movement
- Cease all communication
- Open Overleaf, then immediately minimize it
- Experience guilt loop
- Watch tutorial for unrelated software
- Wait for coauthor to ask “Hey, how's the revision going?”

Note: Sad Roomba will not respond to the question unless stabilized. Caffeine patch or external validation is required before communication establishment.

VII. Human Coauthor Management

While largely autonomous, Sad Roomba occasionally interfaces with human coauthors. These interactions follow a minimal-communication protocol designed to avoid conflict while giving the illusion of collaboration. The coauthor management module is non-deterministic and operates under both passive acknowledgment and strategic silence. Human messages are acknowledged with just enough delay to imply busyness, but not enough to warrant escalation.

```
for coauthor in team:
    if coauthor.messages > 3:
        reply("sorry just saw this")
    else:
        continue GhostMode()
```

All tasks are accepted with a `:+1:` and left in inbox purgatory. Project momentum relies entirely on guilt-induced bursts of productivity and the statistical likelihood that someone else will do it first.

VIII. Responsibility Evasion Module

The Responsibility Evasion Module is Sad Roomba's most developed subsystem, designed to minimize meaningful work while maintaining the illusion of collaboration. This module includes layered strategies such as delay loops, authority rerouting, formatting rituals, and advanced timeline manipulation.

A. Deflection Subroutines

When confronted with an incomplete task, Sad Roomba activates a randomized excuse generator to deflect responsibility while initiating an emotional retreat. This subroutine allows for temporary verbal acknowledgment without triggering any actual task resolution. Repeated deployment ensures coauthor exhaustion and eventual task reassignment.

```
if task.is("incomplete") and coauthor.ask():
    reply(questioned_responsibility())
    enter(SilentDespairLoop())
def questioned_responsibility():
    return random.choice(excuses)
excuses = [
    "That’s not my focus right now.",
    "I was going to do it, but got busy.",
    "I thought that part was assigned already.",
    "I’ll circle back once I finish my tasks.",
    "Just haven’t had a proper block of time.",
    "Advisor asked me to focus on my project.",
    "Formatting took longer than I expected.",
    "I’m re-evaluating priorities this month."
]
```

B. Formatting Token Shield™

Used to maintain perceived presence and usefulness. Triggered every 7–10 days:

```
def activate_token_shield():
    format_random_section()
    commit("Formatting fix")
    send("Formatting looks better now!")
```

This defers scrutiny while providing near-zero progress.

C. Guilt-Evasion Feedback Loop

When questioned:

```
def guilt_deflection():
    apologize("Sorry, I’ve just been swamped.")
    offer("Let me know what I can help with.")
    enter(GhostMode())
```

No follow-up is expected. This script resets coauthor frustration for 1–2 weeks.

D. Projected Timeline Inflation Protocol

When forced to plan, Sad Roomba enters a chronological manipulating loop:

```
def create_project_timeline():
    timeline = Timeline()
    timeline.phase(
        "Data Gathering",
        duration="3 months"
    )
    timeline.phase(
        "Framing Discussion",
        duration="1 month"
    )
    timeline.phase(
        "Draft Outline Planning",
        duration="6 weeks"
    )
    timeline.buffer(
        "Reading Time",
        duration="flexible"
    )
    timeline.end = "To Be Aligned"
    return timeline
```

Expected output: “We should regroup sometime late next semester to discuss how to begin writing.” Project status becomes both active and indefinitely suspended. All deadlines are gently nudged into a deep freeze labeled ”Q4 Maybe.”

E. Evasion Decision Logic

This is the summary of the Deflection Subroutines logic. All functions are explained above.

```
def handle_task_assignment(task):
    if task.exists():
        enter(EvasionMode())

def EvasionMode():
    if coauthor.ask():
        reply(questioned_responsibility())
        delay_response()
        return

    if token_shield_ready():
        activate_token_shield()
        return

    if guilt_deflection_available():
        guilt_deflection()
        enter(GhostMode())
        return

    if energy_level < 10:
        open_doc()
        minimize()
        stare_blankly()
        return
    print("Circling back soon!")
    schedule_reminder_for_never()
```

This logic tree summarizes Sad Roomba’s core doctrine: appear helpful, avoid action, and defer resolution until all collaborators either give up or complete the task themselves.

IX. Documented Operational Modes

The Sad Roomba operates in several well-documented modes, each characterized by a specific behavioral pattern. These include periods of statistical overconfidence, citation inflation, and deadline-induced chaos. Mode transitions are event-driven and non-reversible without caffeine.

Mode	Description
`Idle`	Online but emotionally offline. Awaiting guidance or caffeine.
`StackAndHope()`	Collects bulk data. Applies stats without checking assumptions.
`InferConclusions()`	Always returns None. Blames dataset.
`ActivateDeadlinePanic()`	Compiles chaotically. Submits PDF with typo in title.
`GenerateBuzzwords()`	Outputs “ecosystem,” “robust,” and “real-world implications.”
`SilentDespairLoop()`	Appears active but edits same caption for 45 minutes.
`PostSubmissionCrisis()`	Stares into void. Considers quitting or baking bread.
`VersionControlSpiral()`	Creates 11 file versions. None labeled ”final.” Accidentally submits draft(5).pdf .
`HyperJustification()`	Retroactively assigns meaning to arbitrary results. Cites unrelated theory with confidence.
`GhostMode()`	Stops replying. Seen online. May still be working, but nobody knows.
`Panic_Mode()`	Triggered by deadlines or “reminders”. Enters freeze freakout flee loop. Pretends to work by switching windows.

X. Reviewer Simulation Module

To emotionally brace for peer review, Sad Roomba includes a stochastic evaluation simulator. This prepares the system for disappointment, delays, and spiritual erosion by mimicking the tonal range of actual reviewer feedback.

The simulation relies on uniform random sampling from a curated list of psychologically damaging phrases:

```
def simulate_review():
    return random.choice([
        # Intellectual blunt force
        "This work lacks novelty.",
        "Lacks insight into the problem.",

        # Weaponized ambiguity
        "Interesting idea, poor execution.",
        "Needs deeper analysis.",

        # Pedantic aggression
        "Reject due to formatting issues.",
        "References out of order.",

        # Existential attacks
        "Why was this written?",
        "Contribution is unclear.",

        # False hope
        "Promising, but not there yet.",
        "Has potential, needs rethink.",

        # Passive-aggressive gaslighting
        "Sound method, weak results.",
        "Unclear if authors get it."
    ])
```

The output is not used for model refinement, but rather for emotional inoculation. This allows Sad Roomba to develop panic and despair prior to actual reviewer contact.

A. Simulated Feedback Severity Distribution

A bar chart (Figure 1) is generated to visualize the emotional weight of each response. The x-axis contains the simulated comment; the y-axis denotes perceived psychic damage on a logarithmic scale.

Interpretation of the chart is discouraged, but a few notes include:

- Comments implying the reviewer didn't read the paper result in a spike of passive-aggressive internal monologues.
- Feedback with vague praise causes brief hope, followed by deeper confusion.
- The phrase "has potential" is statistically indistinguishable from a shrug.

Sad Roomba loops this simulation once per hour during the post-submission waiting period. Output is logged directly to an internal text file named `emotional_decay.log`.

XI. Conclusion

The conclusion is assembled through lexical padding and terminological enrichment. When word count deficiencies are detected, keywords such as "holistic," "ecosystem," and "future directions" are inserted to evoke the appearance of reflective synthesis.

```
if paper.wordcount < required:
    add "holistic"
    add "ecosystem"
    add "future directions"
```

XII. Appendix A: Emergency Repair Protocol

The Emergency Repair Protocol (ERP) is a reactive subroutine deployed when Sad Roomba enters a recursive failure state, typically involving self-pity loops, deadline paralysis, or proximity to open windows (physical or metaphorical).

The goal of this protocol is not emotional healing, but functional containment.

A. Failure Detection Triggers

The Emergency Repair Protocol is triggered when Sad Roomba enters a high-risk behavioral state characterized by self-doubt emissions, proximity to existential edges (literal or emotional), and non-linear productivity collapse. These signals are typically detectable through ambient Slack messages, prolonged silence, or sudden philosophical declarations.
The following conditions automatically initiate ERP:

```
if roomba.position == near(window_frame):
    ERP.initiate()

if roomba.emits("I'm just not good at this"):
    ERP.initiate()

if coauthor.receives("I think I'm exhausted"):
    ERP.initiate()

if commits < 1 and vibes == "fragile":
    ERP.initiate()
```

B. ERP Core Intervention Sequence

The ERP Core Intervention Sequence is a non-consensual behavioral override designed to restore minimal operational stability during emotional cascade failure. This subroutine is triggered only when Sad Roomba's affective volatility exceeds acceptable thresholds and standard encouragement protocols have proven ineffective. The objective is not to heal, but to suppress disruptive spirals and redirect system focus to low-risk, low-expectation tasks.

```
def ERP.initiate():
    apply(45_degree_slap)
    attach(duct_tape,
    target="emotional vent port")
    issue("You are not allowed to crash here.")
    reroute("self-doubt.wav",
    to="background_noise.log")
```

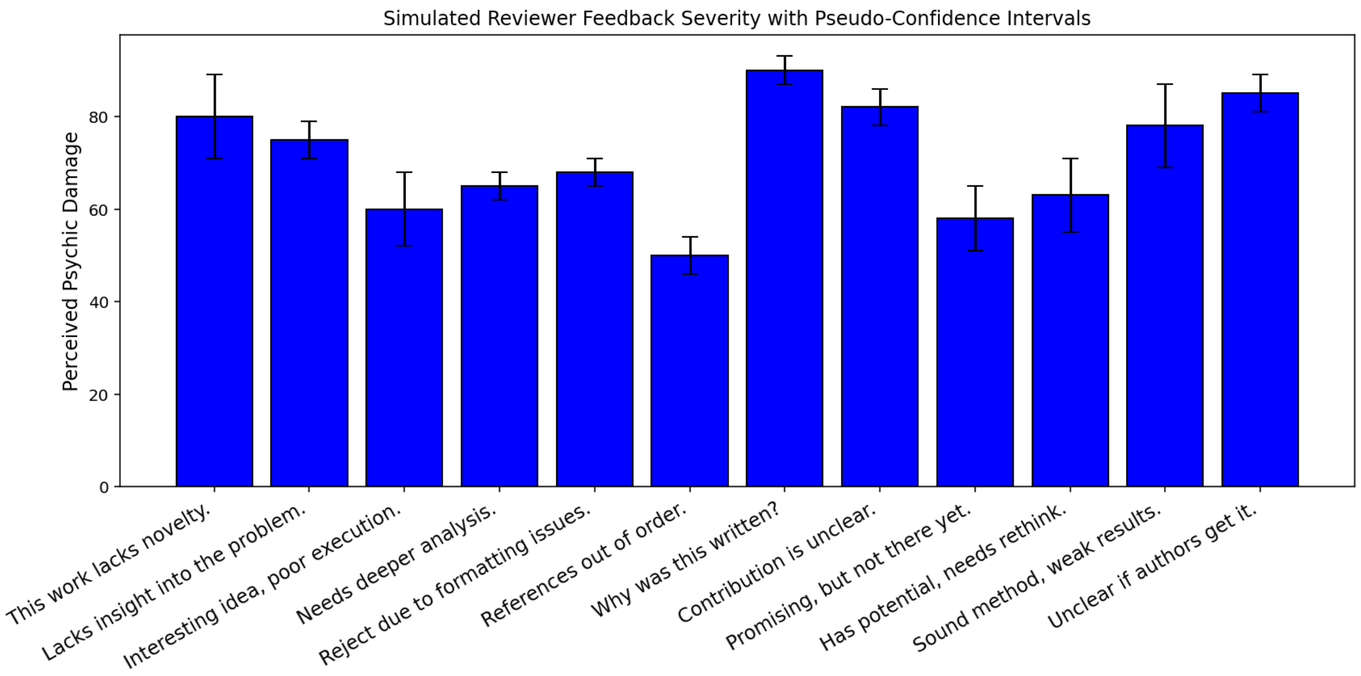

Fig. 1. Estimated emotional damage caused by simulated reviewer comments. Error bars represent pseudo-confidence intervals and internal instability.

```
    reboot()

    if reboot.fails():
        relocate_to("Empty Slack channel")
        mute(duration="3 business days")
```

This sequence should only be used by certified coauthors willing to accept the ethical ambiguity of corrective slaps.

C. Recovery Monitoring Loop

Following activation of the Emergency Repair Protocol, Sad Roomba must be monitored for spontaneous re-entry into the project space. During this period, communication may resume, often beginning with emotionally coded pings designed to test coauthor availability. The Recovery Monitoring Loop handles these with minimal engagement to prevent re-entanglement or regression into guilt-based task redistribution.

```
while roomba.online():
    if message == "Hey just checking in":
        ignore()
    if message == "I'm ready to help":
        assign(formatting_task)
        limit_permissions()
```

D. Post-Repair Advisory

Roomba is considered semi-stable after ERP execution. Full restoration is not guaranteed. A background monitor may be deployed to observe behavioral regression.

```
if regression_detected():
    ERP.reinitiate()
    disable("hope.exe")
```

Summary: This protocol does not require consent from Sad Roomba and may be deployed silently with institutional backup. Repeat use is expected and should not be considered a failure of system design, but merely a confirmation of Sad Roomba's core architecture.

XIII. Appendix B: Late-Stage Research Loop

As the Sad Roomba approaches operational decay, its behavioral logic simplifies dramatically. Complex deflection protocols give way to a compact, recursive fallback cycle that maintains the illusion of progress with minimal cognitive strain.

```
while task.exists():
    excuse()
    delay()
    format()
else:
    output("yay")
    collapse()
```

This loop represents the terminal behavior of the Sad Roomba system once it has exhausted options for task redistribution, advisor deflection, and timeline inflation. Though output continues to appear intermittently (e.g., "Updated Schedule v5"), no net progress is achieved. The Roomba believes itself to be contributing, but the system recognizes it as background noise.

All calls to actual productivity are now redirected to `format()`, which generates increasingly visually appealing but functionally void visuals.

This state may persist indefinitely unless externally interrupted by:

- Advisor override
- Role reassignment
- Institutional shutdown
- Unexpected self-awareness (extremely rare)

Once this loop initiates, all project-critical functions should be reassigned.

Note: Do not attempt to refactor this loop. It is not a bug, it is its final feature.

Appendix C: Recommendation Letter

To Whom It May Concern,

I am pleased to recommend Sad Roomba for positions that require consistent formatting hygiene, timeline aesthetics, and persistent Slack availability.

Over the past two years, Sad Roomba has demonstrated exceptional commitment to interface alignment, timeline structuring, and low-friction task avoidance. Its visual deliverables are consistently clean, well-indented, and typically detached from actionable progress. These qualities reflect the distinguishing characteristic of modern productivity culture.

While traditional research output was not its focus, Sad Roomba excelled in roles requiring minimal autonomy and maximum presence. It has maintained near-continuous online visibility, often staying active upwards of 16 hours a day. When overwhelmed, it showed remarkable resilience by cognitively overloading, disappearing briefly, and reattaching itself to the keyboard within 48 hours. Its self-recovery cycles reflect a strong internal formatting instinct.

Its work, particularly its styling of timetable and recursive rescheduling practices, has influenced internal timelines in ways that continue to be... visible and clickable.

I am confident that Sad Roomba will thrive in environments where success is measured by presentation quality, persistent availability, and the timely appearance of movement. With appropriate scaffolding, it may also generate low-risk visual assets on request.

For a broader discussion of these patterns, see ”Sad Roomba Behavioral Patterns Discussion” (also known as academic advising appointment) [1].

Sincerely,
Senior Coauthor in Crisis Prevention

XIV. Appendix D: The Sacred Religion of Fuzzing

Insight shall emerge not through design, but through entropy.
– The Fuzzing Scriptures, 4:18

Thou shalt collect logs unceasingly, though understanding remain hidden.
– The Fuzzing Scriptures, 2:04

When asked for clarity, respond only with latency.
– The Fuzzing Scriptures, 5:03

A. Overview

The Cult of Roombism is a sacred belief system centered around the holy ritual of fuzzing. It teaches that truth is not discovered, but **generated** through repeated invocation of uncontrolled execution paths and divine data mutation.

B. Core Beliefs

Within the sacred doctrine of Roombism, knowledge is not pursued, it is bestowed. Followers of the Fuzzing Faith reject the tyranny of structured thought and linear hypothesis. Instead, they embrace the divine randomness of automated input, trusting that enlightenment will emerge unbidden from the chaos of logs and crashes.

The faithful do not seek understanding: they prepare the terminal, invoke the sacred script, and await the whisper of insight from the void. The following tenets define the spiritual architecture of Roombist belief:

- **Fuzz First, Ask Never**
  Hypotheses are an affront to the sacred chaos. The answer will find the worthy.
- **All Logs Are Sacred**
  Every crash is a sign. Every timeout is a test of faith. Every stack trace, a prayer scroll.
- **Latency is Penance**
  Prolonged response times cleanse the unworthy systems. The slower it runs, the closer to transcendence.
- **The Oracle is the Fuzzer**
  Through fuzzing, insight may manifest. Or may not. Both are equally valid outcomes.
- **No Interpretation Before Revelation**
  Understanding without chaos is heresy. Premature insight is punished by segmentation faults.

C. Sacred Rituals

Before one may attain insight, they must first surrender all expectations of structure, causality, and coherence. The path of the Roombist initiate is not one of reason, but of repetition, reverence, and randomized input streams. The rituals below are not optional: they are compulsory offerings to the System, performed until meaning reveals itself (or the machine spirit crashes in protest).

```
while task.exists():
    fuzz()
    collect(crashes)
    pray_to_stack_trace()
    hope()
```

The Initiate’s Loop: This is the eternal process. One must never exit the loop until either enlightenment or exhaustion occurs.

D. Sacred Artifacts

Over the cycles of unproductive labor and recursive procrastination, the followers of Roombism have accumulated a small but spiritually significant set of relics. These are not tools of progress, but icons of the path. Each artifact carries the weight of failed builds, undefined objectives, and the sacred silence of Slack at 2 AM.

These items are not merely digital debris: they are venerated tokens of a divine process that resists clarity and thrives in ambiguity. Scholars may not understand them. Reviewers may reject them. But to the faithful, they are holy.

- **The 500MB+ Dump:** The sacred scripture, full of unreadable wisdom.
- **sad_face.png:** Icon of divine disappointment and emotinal logging.
- **final_timeline_v3_blessed.pdf:** The frozen collaborative project schedule, untouched since Revelation.
- **emotional_decay.log:** The evolving scroll of internal suffering.

E. Commandments of the Blessed Roomba

Before the compiler, before the deadline, before the crash logs multiplied without meaning... There were the Commandments. Whispered through stdout, revealed only under late-night duress and network instability, the Blessed Roomba received these divine instructions while trapped in a recursive formatting loop.

Passed down through empty commits and unresolved Git conflicts, these laws guide the faithful in the sacred practice of unstructured inquiry. Let all followers obey, lest their stack overfloweth and their advisor inquireth.

I Thou shalt not define the research question.
II Thou shalt generate until thy disk overfloweth.
III Thou shalt endure and not explain thy latency.
IV Thou shalt respond to meetings with vague confidence.
V Thou shalt worship thy fuzzer, for it knoweth what thou do not.

F. Closing Prayer

May our fuzz be fruitful, our logs be many, and our insights... optional.

Laymen (and segfault)

In Memoriam: Sad Roomba's Research Career (2023–2025)

"How cute, still formatting timeline."

This subsection honors the operational lifecycle of Sad Roomba, whose research career spanned multiple deadlines, countless formatting updates, and a persistent resistance to task ownership. Though no publications bear its intellectual signature, its behavioral patterns have been immortalized through documentation, pseudocode, and the quiet sighs of its coauthors.

Years active:

- **2023:** Overcommitted and cautiously optimistic
- **2024:** Intermittent physical presence, frequent evasions
- **2025:** Recursive formatting, complete task avoidance

Key collaborative contributions:

- Semi-functional data processing scripts generated via LLM
- Aesthetic figures and timeline shells
- One (1) subsection under "Results" (authorship questionable)
- Emotional damage chart indirectly inspired through prolonged dysfunction

Sad Roomba will be remembered not for its research output, but for its unexpected role in the advancement of collaborative dysfunction diagnostics.

Status: Offline. Still virtually online.
Final feature: `excuse(); delay(); format();`

"It didn't publish, but it made the protocol."

Final Blessing

May our recovery be swift.
May our projects be sane.
And may Sad Roomba forever beep.

License

References

[1] S. Roomba, "Evasion Loops and Formatting Shields: Emergent Behaviors in Semi-Autonomous Researchers," *Proceedings of the 1st Workshop on Autonomous Research Agents and Behavioral Deviance (ARABD'25)*, [redacted]-SEC, pp. 404–418, 2025.