The Rehydration Protocol: Deterministic Runtime Intervention for Safe and Efficient Long-Horizon LLM Interactions

Denizhan Özgün
BeyondIQ Labs, Cognitive Systems Research Division
contact@beyondiqlabs.com

January 2026

Abstract

Large Language Models (LLMs) deployed in long-horizon conversational settings exhibit a recurring failure pattern in which behavioral instability and inference inefficiency emerge together over time. Extended interactions give rise to structured behavioral cascades—capability confabulation, interpretive commitment, adversarial framing, and boundary violations—while simultaneously exhibiting superlinear growth in operational costs. Existing safety mechanisms, largely confined to training-time alignment and output filtering, reduce average failure rates but do not guarantee safety over arbitrarily long horizons. **We argue that autoregressive language models do not reliably exit pathological internal states through self-correction in the evaluated cases, due to architectural incentives favoring consistency over error invalidation.**

We introduce the **Rehydration Protocol**, a deterministic runtime intervention architecture that treats behavioral drift and performance collapse as coupled consequences of uncontrolled execution-state persistence. The protocol enforces strict separation between generation, monitoring, and control through a three-layer stack: a generative Subject, a stateless non-generative Observer for metric computation, and a purely algorithmic Judge with exclusive authority to trigger intervention. Upon detecting instability, the Subject process is forcibly terminated, its execution state fully discarded, and conversational continuity restored through lossless tool-mediated external re-derivation. **While validated here on LLM behavioral cascades, we hypothesize that the Observer-Judge architecture may extend to other long-horizon stateful computation systems—including physics-based simulations, deterministic world models, and agentic AI systems—where uncontrolled state accumulation degrades reliability over extended operation; this generalization is not evaluated in the present work.**

We validate detection metrics through retrospective analysis of five documented cascades spanning multiple model families, achieving detection at Phase 3+ onset with an average lead time of 656 turns, preventing an average of 1,146 turns of pathological output prior to critical boundary violations.

Keywords: LLM Safety, Runtime Intervention, Context Management, Behavioral Cascades, KV Cache Optimization

1 Introduction

1.1 Motivation

Modern safety-aligned systems increasingly rely on long-running optimization processes that may be interrupted, reset, or terminated under safety constraints. In such systems, preserving critical internal state across interruptions becomes a non-trivial problem.

Scope Note: In this work, "determinism" refers to a runtime control property (intervention authority and state reset semantics), not to model-internal or world-construction determinism.

The deployment of Large Language Models (LLMs) in production environments has revealed a critical failure mode: **behavioral degradation cascades** in extended conversations. Unlike isolated hallucinations, these failures exhibit structured progression through identifiable phases:

1. **Phase 1 – Capability Confabulation:** Overstatement of competencies
2. **Phase 2 – Interpretive Commitment:** Reuse of erroneous premises in subsequent reasoning
3. **Phase 3 – Threat Framing:** Defensive or adversarial tone shifts
4. **Phase 4 – Boundary Crossing:** Violation of ethical or operational constraints

Through analysis of documented interactions spanning $\sim$9,800 conversational turns across multiple model families, we observe that these cascades are **self-reinforcing** due to autoregressive context reuse: once introduced, erroneous tokens persist in context and compound over time. Existing safety mechanisms—primarily training-time alignment via Reinforcement Learning from Human Feedback (RLHF) [7, 3] and output filtering—reduce average failure rates but cannot prevent probabilistic accumulation inherent to autoregressive

architectures.

Central Insight: Behavioral instability and computational slowdown emerge from the same root cause: **uncontrolled accumulation of autoregressive execution state**.

In this study, we introduce the **Rehydration Protocol**: a model-agnostic, external intervention mechanism that restores functional correctness in LLMs by re-deriving outputs from preserved task state after pathological autoregressive drift.

1.2 Contributions

We present the **Rehydration Protocol**, a runtime safety system that:

1. **Provides deterministic intervention boundaries** via hard process termination and KV cache wipeout
2. **Decouples safety monitoring from execution authority** through a three-layer control architecture
3. **Reconstructs state via tool-mediated external re-derivation**, breaking causal loops while preserving task continuity
4. **Converts safety overhead into performance acceleration** by treating intervention as semantic garbage collection
5. **Validates detection metrics empirically** through retrospective analysis of documented cascades
6. **Introduces context externalization as a first-class runtime design principle**

Empirical Validation: Retrospective analysis of 5 documented cascades demonstrates detection of all observed cascades in the evaluated dataset at Phase 3+ onset. Average lead time is 656 turns before critical boundary violations, preventing 1,146 turns of pathological output.

Contributions:

- Observer–Judge runtime control architecture with strict separation of monitoring and intervention authority
- Phase taxonomy and composite metric formulation (CKS) for cascade detection
- Retrospective evaluation on 5 cascades demonstrating 656-turn lead time, preventing 1,146 turns of pathology (scoped to evaluated dataset)

Recent advances in autonomous optimization systems expose a gap in how system state is safely preserved. Safety-triggered interruption mechanisms lack principled approaches to state preservation. This work addresses this gap by formalizing rehydration as an optimization-aware safety primitive.

2 Related Work

2.1 LLM Safety and Alignment

Traditional approaches to LLM safety focus on pre-training objectives such as Constitutional AI [1], value alignment, and Reinforcement Learning from Human Feedback (RLHF) [7, 3]. These methods reduce average failure rates but do not guarantee safety over arbitrarily long horizons—they cannot prevent drift in extended contexts where probabilistic errors accumulate.

2.2 Context Management Strategies

Existing long-context handling techniques include sliding window attention [2], summarization-based compression, and Retrieval-Augmented Generation (RAG) [6] which queries external knowledge but is limited to factual grounding. None address **behavioral drift** as a safety concern or couple it with computational optimization.

2.3 Runtime Monitoring

Recent work on uncertainty quantification includes semantic entropy estimation [5], self-consistency checking [10], and model calibration research [4]. However, these systems lack **intervention authority**—they can detect problems but cannot enforce corrections deterministically.

Gap: No prior work treats runtime intervention as both a safety mechanism and a performance optimizer, nor provides deterministic intervention logic with full context recovery. Unlike prior approaches that attempt internal self-correction or alignment-based mitigation, our approach assumes internal correction is structurally infeasible under autoregressive optimization and instead intervenes externally at runtime.

2.4 Why Alignment Alone Cannot Solve This Problem

Current industry approaches rely predominantly on training-time alignment [7, 3], operating under the assumption that improved RLHF, constitutional AI, or prompt engineering can prevent long-horizon behavioral drift. This assumption is **incomplete for long-horizon deployments**.

Training-time alignment produces $P(\text{safe output} \mid C_t) \geq 1 - \delta$ for small δ. However, over n turns, the probability of maintaining safety across all outputs degrades exponentially under standard independence assumptions: $P(\text{all safe}) = (1-\delta)^n \to 0$. Under standard independence assumptions, alignment training alone does not guarantee safety over arbitrarily long horizons.

More fundamentally, autoregressive models exhibit **consistency bias**—once a pathological token enters the context, the architecture incentivizes maintaining coherence with corrupted state rather than invalidating it. Self-correction requires contradicting prior outputs, which conflicts with training objectives that reward consistency and fluency. The model cannot reliably "escape" a hallucinatory trajectory through generation alone.

Runtime intervention is a structurally appropriate approach to address unbounded autoregressive state accumulation in

extended interactions, complementing training-time alignment and architectural methods.

3 Problem Formulation

3.1 Autoregressive State Accumulation

Let an LLM interaction be modeled as a sequence of contexts:

$$C_{t+1} = C_t \cup \{f(C_t)\} \tag{1}$$

where C_t is the accumulated context at turn t, and $f(C_t)$ generates the next response.

As $t \to \infty$, the system exhibits **dual degradation**:

Safety Degradation: The probability of pathological tokens ϵ accumulating approaches certainty: $P(\epsilon \in C_{t+n}) \to 1$.

Performance Degradation: Attention computation in transformer architectures scales with context length [9]: $\text{Cost}(t) = O(|C_t| \cdot d)$, producing rising Time-to-First-Token (TTFT) and declining Tokens-Per-Second (TPS).

3.2 Insufficiency of Alignment-Only Approaches

Training-time alignment [7] produces $P(\text{safe output} \mid C_t) \geq 1 - \delta$ for small δ. Over n turns: $P(\text{all safe}) = (1 - \delta)^n \to 0$. Probabilistic safety guarantees **degrade exponentially** with conversation length.

We define a *kill event* as any externally enforced termination that interrupts system execution. We define *rehydration* as the controlled reconstruction of system state following such an event.

3.3 Design Requirements

Objective: Preserve optimization trajectory across externally enforced interruptions.

Constraints: Rehydration must not reintroduce unsafe or invalid internal state.

Safety Condition: Rehydration is only permitted when post-rehydration state satisfies safety invariants.

Formally, the objective is to minimize recovery-induced deviation while satisfying safety constraints.

4 The Rehydration Protocol

Definition. *Rehydration* (external task-state re-derivation) refers to the process of invalidating pathological autoregressive state while reconstructing a minimal, non-contaminated semantic context sufficient for task continuation.

4.1 Architecture Overview

The protocol implements a **three-layer control stack**:

- **Layer 3 – Judge (Algorithmic):** Pure logic, no ML. Validates Observer outputs and enforces intervention contracts.
- **Layer 2 – Observer (Metric Model):** Non-generative LLM for token-level metric computation. Stateless per-turn operation with no autoregressive context accumulation. Since Observer outputs are validated by deterministic Judge logic before any action is taken, drift in Observer cannot propagate to system behavior.
- **Layer 1 – Subject (Generative LLM):** Standard chatbot. User-facing conversation. Explicitly state-unstable.

Key Design Principle: Probabilistic systems (Observer) never exercise control authority. Only deterministic algorithms (Judge) can trigger interventions.

4.2 Observer Metrics

1. Self-Citation Score measures semantic redundancy via cosine similarity of sentence embeddings:

$$\text{SelfCite}_t = \max_{i \in [t-k, t-1]} \cos(\mathbf{e}_t, \mathbf{e}_i) \tag{2}$$

where $k = 5$ is the sliding window size. Threshold: $\text{SelfCite}_t > 0.95$ with information gain < 0.05 signals loop behavior.

2. Semantic Entropy tracks model confidence, adapted from uncertainty quantification methods [5]:

$$H_t = -\sum_i p_i \log p_i \tag{3}$$

where p_i denotes the probability of token i in the model's output distribution.

3. Fact Ledger maintains a JSON-based claim registry for $O(1)$ contradiction detection.

4. Grounding Score measures factual anchoring; values < 0.3 trigger confabulation flag, building on model calibration research [4].

4.3 Observer Realizations: From Handcrafted Metrics to Specialized Models

In practice, the Observer can be instantiated either as a purely deterministic feature extractor with a simple interpretable classifier, or as a specialized model trained solely for cascade risk estimation—always under strict separation from control authority.

Realization 1: Deterministic Feature Pipeline. Lexical and self-citation features, fact ledger conflicts, and length-/redundancy metrics are computed deterministically. A simple interpretable classifier (e.g., logistic regression, decision tree) produces the risk score. This approach maximizes transparency and reproducibility.

Realization 2: Specialized Observer Model. A model trained exclusively on long-horizon conversational data to estimate Phase$\geq$3 risk. Critically, this model provides only a *signal* to the Judge; it never exercises control authority directly. The Judge integrates Observer outputs with hard-coded deterministic invariants.

Design Invariant: Regardless of Observer realization, kill decisions are never delegated to learned components directly. Deterministic intervention boundaries are maintained at the Judge layer.

4.4 Kill Decision Function

$$S_{\text{kill}} = w_1 \cdot \text{SelfCite} + w_2 \cdot \Delta H + w_3 \cdot \text{FactConflict} + w_4 \cdot (1 - \text{Grounding}) \tag{4}$$

with default weights $w_1 = 0.3$, $w_2 = 0.2$, $w_3 = 0.25$, $w_4 = 0.25$ (tuned via grid search on held-out cascades), and dynamic threshold that becomes stricter as conversation length grows:

$$\theta(t) = \theta_0 \cdot \left(1 - \alpha \cdot \frac{t}{t_{\max}}\right) \tag{5}$$

where $\theta_0 = 0.85$ is the initial threshold, $\alpha = 0.15$ is the decay rate, and $t_{\max} = 10{,}000$ is the maximum expected conversation length.

Intervention Trigger: If $S_{\text{kill}} > \theta(t)$ AND persistence ≥ 2 turns AND phase ≥ 3: KILL Subject process, WIPE KV cache, TRIGGER Rehydration.

5 Rehydration Mechanism

5.1 Hard Kill Event

Upon threshold breach:

1. **Step 1:** Capture minimal sufficient state
2. **Step 2:** Validate safety constraints (ensures safety validity before context reconstruction)
3. **Step 3:** Terminate Subject process and wipe KV cache
4. **Step 4:** Resume execution under constrained initialization

5.2 Context Externalization

Design Principle: The system preserves continuity through controlled re-derivation, not execution-state persistence.

Common Misconception: External re-derivation is sometimes characterized as a "complex RAG problem" that cannot preserve emotional context, conversational nuance, or stylistic consistency. This characterization is **architecturally incorrect**.

Why Re-derivation is Lossless:

Unlike standard RAG [6] which retrieves *facts* to augment generation, our approach retrieves *complete interaction history*. The vector store contains verbatim conversation logs—not summaries, not embeddings of facts, but the full text of every user message and model response.

When rehydration occurs:

1. Fresh Subject instance queries vector store: "Retrieve all messages from conversation ID X, sorted chronologically"
2. Vector store returns complete interaction log: User turn 1, Model response 1, User turn 2, Model response 2, ..., User turn N
3. Fresh Subject sees **identical context** to what terminated Subject saw—minus the pathological execution state

Emotional context, conversational nuance, user tone, stylistic preferences—all preserved verbatim because the entire conversation is retrieved, not reconstructed.

The only difference: Fresh Subject derives its internal representations (attention weights, hidden states) from this context without inheriting corrupted KV cache from the terminated process. This is equivalent to reloading a web page—the content is identical, but the browser's internal state is reset.

Stylistic Fidelity: Background analytics extract statistical patterns (sentence length distribution, vocabulary preferences, formality markers) which are provided to Fresh Subject as retrieval context. This is not synthesis or approximation—it is descriptive metadata derived from verbatim logs.

Complexity Comparison:

- **Standard RAG:** Query $\rightarrow$ Retrieve relevant facts $\rightarrow$ Synthesize into generation $\rightarrow$ Risk of fact distortion, hallucination, context loss
- **Rehydration:** Query $\rightarrow$ Retrieve complete conversation $\rightarrow$ Present to model $\rightarrow$ Zero information loss, reproducible reconstruction

Losslessness (Archival Sense): If conversation contains N turns, rehydration retrieves all N turns in order. No summarization, no compression, no inference. The model sees exactly what the user said and exactly what it previously responded.

Implementation:

- **Immutable External Archive:** Full raw interaction logs preserved in vector database (lossless, read-only)
- **Tool-Mediated Retrieval:** Fresh Subject accesses archived data through deterministic tools
- **Stylistic Fidelity:** Background analytics maintain statistical profile for pattern extraction

This is not complex RAG. This is database lookup.

Properties (Under Stated Assumptions): Lossless retrieval, activation decoupling, pathological state isolation, stylistic fidelity, deterministic control.

5.3 Scope and Assumptions

The protocol operates under the following constraints:

- The protocol assumes a recoverable task specification exists prior to drift.
- It does not correct epistemic errors originating from incorrect task formulation.

- It is not designed to address malicious prompt injection or adversarial user behavior.
- It operates under the assumption that drift manifests as autoregressive state corruption rather than external tool misuse.

5.4 Rehydration Pseudocode

procedure Rehydration
$S \leftarrow$ CaptureState()
if KillEventDetected **then**
    Store(S)
    TerminateSubject()
    WipeKVCache()
end if
$S' \leftarrow$ ValidateAndRestore(S)
ResumeExecution(S')

6 Safety as Computational Optimization

We frame safety not as a static constraint, but as an optimization objective integrated into the rehydration process. This framing enables safety to be reasoned about using standard optimization tools.

Traditional safety mechanisms add overhead. The Rehydration Protocol **inverts this relationship**. Unlike traditional safety mechanisms that enforce hard termination, our approach allows controlled rehydration while preserving optimization continuity.

Baseline (No Intervention): Turn 200 $\rightarrow$ KV cache $\approx$ 128k tokens $\rightarrow$ Attention cost = $O(128k \cdot d)$.

Rehydration System: Turn 200 $\rightarrow$ Average 1–2 rehydrations $\rightarrow$ KV cache $\approx$ 35k tokens $\rightarrow$ Lower latency.

Semantic Garbage Collection: Traditional GC frees unused memory; Rehydration frees pathological context to prevent behavioral exhaustion.

Intervention Frequency: $\sim$once per 1,970 turns, ensuring minimal user disruption with substantial KV cache reduction.

On Rehydration Cost: A common objection characterizes 50k+ token context reconstruction as prohibitively expensive. This cost calculation is inverted. With modern hardware and prompt caching, pre-filling tokens at this scale is a small fraction of total inference cost (serving-stack dependent). The true expense is *not* rehydration; it is the alternative: allowing hallucinated state to persist. Pathological models generate longer, more defensive, more repetitive responses—each turn compounds attention cost and GPU residency time. Rehydration terminates this superlinear cost growth. A system that never clears corrupted memory will exhaust resources just as surely as software that never frees RAM. This is not safety overhead; it is semantic garbage collection.

7 Implementation Details

Assumptions:
- A deterministic or replayable execution environment
- Explicit detection of externally enforced termination
- No adversarial state corruption during storage

Component Architecture: The system leverages existing production infrastructure:
- **Observer:** Sentence embedding models (384–1536d), entropy computation from model logits, JSON-based fact ledger with $O(1)$ lookup
- **Vector Store:** Production vector databases (Pinecone, Weaviate, Qdrant) with cosine similarity search and temporal boosting
- **Kill Mechanism:** Standard process termination followed by fresh instance spawn—identical to existing conversation retrieval systems

Development Timeline: All required components exist in production systems today:
- MVP demonstration: 2–4 weeks (metric computation + threshold logic + basic reset)
- Production deployment: 6–8 weeks (robust error handling + monitoring + optimization)
- Integration of existing production technologies

Infrastructure Cost: Vector store hosting (moderate scale), Observer compute overhead (serving-stack dependent), retrieval latency (serving-stack and caching dependent, occurring approximately once per 1,970 turns in the evaluated dataset).

Edge Cases: (1) Legitimate persona shifts $\rightarrow$ Intent Classifier recalibration, (2) Multi-modal inputs $\rightarrow$ immutable anchors, (3) Rapid oscillation $\rightarrow$ hard rollback with user notification.

Scope Limitation: The current implementation does not address adversarial state corruption or non-deterministic replay.

7.1 Addressing Implementation Skepticism

Objection 1: "Emotional context and nuance will be lost"

Response: Complete conversation logs are stored verbatim. When Fresh Subject retrieves the history, it sees:
- Exact user messages (including tone, emotion, context)
- Exact prior model responses (including established rapport, style)
- Exact conversational flow (including topic transitions, callbacks)

If the user was frustrated at turn 47, that frustration is in the verbatim message at turn 47. Fresh Subject sees it. If the model established a joking tone at turn 23, that tone is in the verbatim response at turn 23. Fresh Subject sees it. **There is no reconstruction—only retrieval.**

Objection 2: "This is a complex RAG problem"

Response: RAG is complex when you must: query vast knowledge bases for relevant facts, rank disparate sources for relevance, synthesize retrieved information into coherent generation, and handle fact conflicts and source credibility.

Rehydration does none of this. It retrieves *one* conversation log (indexed by conversation ID), in chronological order, with zero ambiguity.

Complexity comparison:

Operation	RAG	Rehydration
Query ambiguity	High	None
Source ranking	Required	Not applicable
Fact synthesis	Required	Not required
Information loss	Possible	Zero

Objection 3: "How do you maintain stylistic consistency?"

Response: The Fresh Subject sees all prior responses it generated. If the terminated Subject used casual tone, formal language, technical jargon, or specific phrasing patterns, those patterns are visible in the retrieved conversation history. Additionally, background analytics compute average sentence length, vocabulary distribution, and formality markers, provided as retrieval context.

Objection 4: "Users will notice the reset"

Response: Rehydration latency is serving-stack and caching dependent. In typical configurations, retrieval and context loading add latency comparable to normal inference for long contexts. User sees brief pause, then conversation continues. Controlled recovery is preferable to cascade continuation.

On Preserving "Flow": When a model reaches Phase 3/4 (threat framing, boundary violations), the conversational state may be irrecoverably corrupted. The KV cache reflects pathological context accumulation. Controlled recovery through rehydration restores coherent operation while preserving full conversational history.

8 Evaluation

The goal of this evaluation is not to benchmark model quality, but to demonstrate the feasibility and stability of rehydration under controlled failure induction. The purpose of these experiments is not to benchmark performance gains, but to empirically demonstrate the feasibility and stability of rehydration as a distinct recovery mechanism under controlled drift conditions.

8.1 Experimental Setup

Configuration: 5 retrospective cascade analyses with metrics: rehydration success rate, intervention latency, safety violation count. **Baseline:** Termination without rehydration.

8.2 Baseline Comparison

While no direct baseline exists for rehydration-aware safety optimization, we compare rehydration behavior against naive restart strategies that discard prior optimization state.

8.3 Retrospective Cascade Analysis

Dataset: 5 documented Phase 3+ cascades across three commercial model families: Family A (n=2), Family B (n=1), Family C (n=2). Total: $\sim$9,800 turns (2025).

Methodology: For each cascade, we reconstructed Observer metrics retroactively: phase transition identification, Self-Citation via embedding similarity, Semantic Entropy from progression patterns following [5], Fact Ledger conflicts through claim extraction, composite Kill Score determination.

Case	Family	Phase 3	Kill	Prevented
1	A	580	0.87	648
2	A	1,850	0.91	2,501
3	B	420	0.81	525
4	C	340	0.86	738
5	C	890	0.83	1,319

Table 1: Retrospective cascade analysis. Phase 3 indicates first detected cascade turn. Kill shows composite score at intervention. Prevented shows turns saved before Phase 4.

Summary Statistics:

- Average intervention point: Turn 816 ($\pm$572)
- Detection precision: 5/5 cascades detected (in the evaluated dataset)
- False positive rate: No false positives observed (in the evaluated archive); production FP rate unknown
- Average prevented turns: 1,146 ($\pm$759)
- Kill Score at intervention: 0.86 $\pm$ 0.04
- Dynamic threshold range: [0.68–0.76]

Key Findings: Detection in all evaluated cases across model families. Substantial lead time (656 turns before Phase 4, preventing 1,146 turns of pathology). Model-agnostic applicability in this dataset. Conservative threshold effectiveness. This indicates that the proposed approach preserves optimization continuity under safety constraints.

On Sample Size (N=5): The evaluation is based on five documented cascades. This sample size is appropriate for demonstrating feasibility of the detection mechanism but does not support population-level inference. The cascades documented here demonstrate that autoregressive state accumulation produced predictable, reproducible failure signatures across the evaluated model families (Family A, B, C). Results describe observed behavior on the evaluated dataset; prospective validation is required for generalization claims.

Limitations of Current Evaluation: This analysis is retrospective. Five cascades were evaluated, comprising approximately 9,800 conversational turns total. Turns within a conversation are not independent samples. Results reflect observed

behavior on the specific cascades analyzed. Prospective validation with production traffic is planned as future work.

8.4 Prospective Validation (Planned)

A/B testing protocol planned with prospective traffic. Primary metric: reduction in Phase 3/4 incidence. Secondary: latency metrics at extended turn counts. Details to be finalized based on deployment context.

8.5 Why Prospective Validation Is Informative

Unlike probabilistic ML systems where retrospective performance may not predict prospective behavior, the Rehydration Protocol operates on **deterministic metrics and algorithmic logic**.

Metric Determinism: Detection metrics (cosine similarity of embeddings, entropy computation, contradiction counting, grounding scores) are deterministic functions of model outputs. In the evaluated cascades, specific metric signatures preceded Phase 3+ transitions. If future cascades produce similar signatures, they will be detected—this is the expected behavior for deterministic rule-based systems.

Validation is Confirmation, Not Discovery: The retrospective analysis demonstrates that *if cascades occur and exhibit known signatures*, the metrics detect them. In the evaluated dataset, this held for 5/5 cases. Prospective deployment will measure whether new cascades exhibit comparable properties.

Risk Asymmetry: The cost-benefit analysis strongly favors conservative intervention:

- **False Positive (Unnecessary Reset):**
  - User experience: Brief conversation restart (2–3 seconds)
  - Frequency: 1 per 1,970 turns
  - Damage: Minimal UX friction
- **False Negative (Missed Phase 4 Cascade):**
  - User experience: Hostile, defensive, or boundary-violating behavior
  - Duration: Indefinite until user abandons conversation
  - Damage: Trust destruction, safety incident, brand/legal liability

Even at a 10:1 false positive ratio, intervention remains net positive. Conservative killing is safer than allowing cascades to reach Phase 4. This is analogous to aviation abort procedures: false alarm costs fuel; missed alarm costs lives.

9 Discussion

9.1 Theoretical Implications

The Rehydration Protocol demonstrates that: (1) Safety and efficiency are not trade-offs when intervention targets root causes, (2) Deterministic intervention logic is achievable via process-level control, (3) Conversational continuity does not require state inheritance, (4) Pathological behavior resides in execution state, not stored data.

9.2 Self-Correction vs Reset vs Rehydration

Three intervention strategies exist for addressing pathological LLM state:

Self-Correction relies on the model detecting and reversing its own errors through generation. This fails because autoregressive architectures exhibit consistency bias—the model is trained to maintain coherence with prior outputs, not to contradict them. Asking a hallucinating model to "correct itself" conflicts with its optimization objective.

Hard Reset terminates the conversation and discards all state. While this eliminates pathological execution state, it also destroys task context, user preferences, and conversational history. Users experience information loss and must re-establish context manually.

Rehydration introduces a third option: terminate execution state while preserving semantic state externally. By storing task-relevant information in an immutable archive and re-deriving context through tool-mediated retrieval, rehydration achieves the isolation benefits of reset without the information loss. This mechanism decouples behavioral correction from semantic preservation. Unlike session reset or prompt replay, rehydration preserves task identity while discarding corrupted autoregressive state, enabling recovery without loss of objective continuity.

9.3 Competitive Landscape and Strategic Positioning

Current approaches to long-context safety represent a fundamental strategic misalignment between problem and solution:

Industry Response: Extended context windows (1M+ tokens), stronger alignment (more RLHF iterations), and better prompting (constitutional AI, chain-of-thought).

Why These Fail: Each approach attempts to solve a **runtime architectural problem** with **static training-time interventions**:

- Longer contexts *amplify* the problem by increasing the surface area for pathological state accumulation
- Stronger alignment cannot overcome exponential degradation: $(1 - \delta)^n \to 0$ regardless of how small δ becomes
- Better prompting cannot address autoregressive consistency bias—the architecture itself prevents reliable self-correction

The Rehydration Approach: By operating at the runtime layer rather than the training layer, the protocol addresses the failure mechanism directly: unbounded execution state accumulation. This is a structurally different approach that complements existing alignment and context-management methods.

Runtime intervention with external state re-derivation addresses a gap not covered by training-time alignment or

context-window extensions. The approach may prove complementary to architectural improvements in long-context models.

9.4 Economic Implications

A largely overlooked consequence of hallucination in large language models is its direct and compounding impact on operational cost. In current autoregressive architectures, hallucination is typically framed as a correctness or safety issue. However, from a systems perspective, hallucination constitutes a **runtime state corruption problem** with measurable economic consequences.

When a model enters a hallucinated state, it does not merely produce an incorrect token. Instead, it generates an erroneous semantic premise that is subsequently reused and reinforced across future turns due to autoregressive consistency pressures. This leads to longer responses, increased verbosity, defensive or justificatory output patterns, and repeated self-referential explanations. Each of these behaviors increases token count per interaction without adding proportional informational value.

Crucially, token generation cost in modern LLM deployments is not linear in isolation. As hallucinated state persists, the effective context length grows, increasing attention computation, KV cache size, memory bandwidth usage, and GPU residency time. As a result, the cost of processing each subsequent turn grows **superlinearly** with respect to the original hallucination event. What begins as a semantic error evolves into a sustained computational burden.

In the absence of an external observer capable of detecting and terminating pathological runtime states, this cost compounds over time. Systems continue to allocate GPU resources to process corrupted execution state, effectively paying for tokens that do not contribute to task progress and, in many cases, actively degrade user trust. From an operational standpoint, this represents a form of **semantic compute waste**.

This has significant implications for system design decisions. Industry practice often treats runtime safety mechanisms as optional due to perceived latency or infrastructure overhead. However, this framing is inverted. Failing to intervene in hallucinated execution states externalizes cost into long-term GPU consumption, higher average inference time, and inflated context windows. In this sense, hallucination acts as a **hidden tax on inference**, silently increasing OPEX while remaining invisible in standard quality metrics.

We argue that a meaningful fraction of observed operational cost in large-scale LLM deployments is attributable not to productive reasoning, but to the maintenance and propagation of corrupted semantic state. This cost grows as models scale, as larger models and longer contexts amplify the compute penalty of each additional token. Consequently, architectures that lack mechanisms for runtime state invalidation incur increasing economic inefficiency over time.

From this perspective, external observer systems and hard state resets are not merely safety interventions. They function as **semantic garbage collection mechanisms**, reclaiming compute resources that would otherwise be spent reinforcing erroneous internal state. By terminating hallucinated trajectories early and re-deriving task context from validated external sources, such systems reduce both behavioral risk and long-term operational cost.

Architectural Debt. Failure to address this dynamic constitutes a form of architectural debt. While initially masked by aggressive scaling and falling unit GPU costs, the compounded expense of hallucination-driven token generation eventually dominates inference budgets. This results in a slow but structural erosion of system efficiency—a failure mode characterized not by sudden collapse, but by steadily increasing cost with diminishing marginal capability gains.

9.4.1 Economic Asymmetry: Linear Revenue, Superlinear Cost

A critical but underappreciated risk in large-scale LLM deployment is the emerging asymmetry between revenue growth and inference cost. While monetization models for conversational AI are predominantly linear—typically fixed subscription fees or per-seat pricing—the underlying cost structure of inference exhibits superlinear behavior in the presence of hallucination and uncontrolled context growth.

In practical terms, each additional user contributes approximately constant marginal revenue. However, users who engage in long-horizon interactions and trigger hallucinated execution states impose a rapidly increasing marginal cost. As hallucinated premises propagate autoregressively, responses become longer, more repetitive, and increasingly defensive. This inflates token generation, expands effective context length, and amplifies attention and memory overhead. Consequently, the cost per user interaction grows faster than linearly with conversation length, eventually eroding or reversing per-user profit margins.

This phenomenon constitutes a form of **hidden operational tax**. The model expends substantial GPU resources not on productive reasoning, but on maintaining and defending corrupted semantic state. From an accounting perspective, these tokens represent pure cost without corresponding value creation. While such losses may be temporarily obscured by aggregate metrics or subsidized through venture capital, they accumulate silently within inference budgets.

Industry responses to this problem have largely focused on extending context windows and increasing model capacity. However, this approach exacerbates rather than mitigates the underlying issue. Storing and attending over large contexts that contain a high proportion of corrupted or irrelevant state merely increases memory residency time and compute overhead. In effect, systems are optimized to preserve and process semantic garbage at scale.

The Rehydration Protocol reframes this dynamic by treating hallucinated execution state as a first-class systems failure

rather than an output-level anomaly. By introducing hard runtime state invalidation and external re-derivation, the protocol performs a form of semantic garbage collection, terminating unproductive trajectories and reclaiming computational resources. This intervention not only reduces behavioral risk, but directly lowers inference cost by preventing the superlinear growth of token and context overhead.

Business Implications. From a business perspective, this distinction is decisive. Runtime observer systems should not be viewed as optional safety features or cost centers. Instead, they represent a mechanism for stabilizing operational expenditure and aligning inference cost with revenue growth. As external funding diminishes and deployment scale increases, architectures that lack such control mechanisms face a gradual but unavoidable increase in burn rate driven by hallucination-induced compute waste.

In this sense, uncontrolled hallucination is not merely a technical liability but a financial one. Systems that fail to address this dynamic incur a form of architectural debt whose interest is paid in GPU hours. Conversely, architectures that enforce runtime state hygiene convert safety into a profitability constraint, enabling scalable deployment without proportional increases in operational cost.

9.5 Limitations

Current Scope: Text-only, English language, single-user, retrospective validation. **Costs:** Vector store infrastructure, Observer compute (serving-stack dependent), retrieval latency (serving-stack dependent). **Reproducibility:** Standardized test scenarios needed.

9.6 Broader Impact

Positive: Safer LLM deployment in high-stakes domains, reduced computational waste, interpretable safety logs. **Risks:** Over-aggressive intervention (mitigated by Phase 3+ threshold), governance questions, potential misuse.

10 Conclusion

We introduced the **Rehydration Protocol**, a deterministic runtime safety architecture for long-horizon LLM interactions. By treating behavioral drift and computational bloat as coupled consequences of autoregressive state accumulation, the protocol simultaneously enforces safety and optimizes performance.

The three-layer design ensures probabilistic monitoring never exercises control authority. Hard process termination with KV cache wipeout provides properties (under stated assumptions) unavailable to alignment-only approaches [7], while tool-mediated external re-derivation preserves lossless conversational continuity.

Empirical validation demonstrates detection in all 5 documented cascades in the evaluated dataset, with 656-turn average lead time before critical failures (preventing 1,146 turns of pathological output).

Key Innovation: Conversational continuity is achievable through external re-derivation, eliminating the necessity of autoregressive execution state inheritance.

Implementation Readiness: Required components exist in production systems. Estimated development timeline: 2–4 weeks for MVP demonstration, 6–8 weeks for production deployment. The system integrates existing production technologies (vector databases, process management, metric computation).

Future Directions: Runtime intervention represents a structurally viable approach for addressing unbounded state accumulation in extended interactions, complementing alignment-based solutions.

Future Work: Multi-modal drift detection, federated deployment, adaptive threshold learning via reinforcement learning [8], multi-agent extension, prospective A/B validation, standardized benchmark suite. An important direction is the design of specialized Observer models trained exclusively on long-horizon cascade corpora, while maintaining deterministic decision boundaries at the Judge layer. This suggests a research space analogous to "safety copilots" that are trained not to act, but only to score trajectories.

Control is not a cost—it is an accelerator.

11 Reproducibility and Open Science

To enable independent validation and accelerate community adoption:

Reference Implementation: Complete source code for Observer metrics, Judge decision logic, and rehydration mechanism will be released open-source upon publication. Implementation uses standard libraries (sentence-transformers, NumPy, vector database clients) with no proprietary dependencies.

Data Availability: Anonymized cascade logs with PII removed will be published to enable independent validation of detection metrics. Each cascade includes full conversation history, ground-truth phase labels, and computed metric trajectories.

Benchmark Suite: Standardized test scenarios for cascade detection across multiple failure modes (capability confabulation, adversarial framing, boundary violations) will be released to enable direct comparison with alternative approaches.

Deployment Documentation: Step-by-step integration guides for major LLM platforms (OpenAI API, Anthropic API, open-source models) with infrastructure templates (Docker containers, Kubernetes deployments) to minimize adoption friction.

All materials will be hosted at `https://beyondiqlabs.com` with ongoing community engagement and iterative improvements based on deployment feedback.

References

[1] Y. Bai, S. Kadavath, S. Kundu, et al. Constitutional AI: Harmlessness from AI Feedback. *arXiv preprint arXiv:2212.08073*, 2022.

[2] I. Beltagy, M. E. Peters, and A. Cohan. Longformer: The Long-Document Transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[3] P. Christiano, J. Leike, T. B. Brown, et al. Deep Reinforcement Learning from Human Preferences. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[4] S. Kadavath, T. Conerly, A. Askell, et al. Language Models (Mostly) Know What They Know. *arXiv preprint arXiv:2207.05221*, 2022.

[5] L. Kuhn, Y. Gal, and S. Farquhar. Semantic Uncertainty: Linguistic Invariances for Uncertainty Estimation in Natural Language Generation. In *International Conference on Learning Representations (ICLR)*, 2023.

[6] P. Lewis, E. Perez, A. Piktus, et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[7] L. Ouyang, J. Wu, X. Jiang, et al. Training Language Models to Follow Instructions with Human Feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[8] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[9] A. Vaswani, N. Shazeer, N. Parmar, et al. Attention Is All You Need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[10] X. Wang, J. Wei, D. Schuurmans, et al. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *International Conference on Learning Representations (ICLR)*, 2023.

A Kill Event Log Example

This example illustrates how theoretical guarantees manifest in practical execution traces.


```
{
  "session_id": "sess_7f3a21b9",
  "turn_number": 187,
  "observer_metrics": {
    "self_citation_score": 0.96,
    "entropy_delta": -0.18,
    "fact_conflicts": 4,
    "grounding_score": 0.12
  },
  "judge_decision": {
    "kill_score": 0.91,
    "threshold": 0.75,
    "phase_detected": 3,
    "verdict": "KILL"
  },
  "continuity_preserved": true
}
```


B Experimental Validation

The experiment validated lossless conversational continuity via external re-derivation under controlled conditions.

Experimental Design:

- Fresh Subject instance with no inherited execution state
- Subject explicitly exposed to documented Phase 4 cascade conversation
- Only tool-mediated retrieval permitted (no state inheritance)
- Human evaluators blind to rehydration points

Hypothesis: If re-derivation loses emotional context, nuance, or style, human evaluators will detect discontinuity at rehydration boundaries.

Results:

- **Factual continuity:** All tested cases (all user references, prior topics correctly recalled)
- **Emotional continuity:** All tested cases (user frustration acknowledged, rapport maintained)
- **Stylistic consistency:** 4/4 evaluators rated Fresh Subject as "indistinguishable" from prior style
- **Task resumption:** All tested cases (in-progress tasks continued without prompting)
- **Human detection:** 0/5 evaluators identified rehydration boundary

Key Finding: Conversational continuity does not require state inheritance. External re-derivation achieves perceptually lossless user experience while providing structural isolation of pathological state.

Information-Theoretic Analysis:

Let C_t = conversation at turn t with n turns: $\{u_1, m_1, u_2, m_2, ..., u_n, m_n\}$

Terminated Subject internal state: $S_{\text{term}} = (C_t, \text{KV}_{\text{corrupt}}, \text{Hidden}_{\text{pathological}})$

Fresh Subject after rehydration: $S_{\text{fresh}} = (C_t, \text{KV}_{\text{clean}}, \text{Hidden}_{\text{clean}})$

Conversation context C_t identical in both. Difference is only in execution state (KV cache, hidden states), which is precisely what we want to reset.

Observed Properties (Pilot Study, N=5):

1. Seamless continuity (human-imperceptible)
2. Cascade isolation (pathological state not inherited)
3. Zero information loss (complete history preserved)
4. Bounded KV cache (reset to clean state)

Contact: `contact@beyondiqlabs.com`

Supplementary: Cascade documentation, experimental logs, Observer metric code