BIOS: Bootstrap Instruction for Operational Safety

A Meta-Layer Architecture for Ensuring Guardrail Execution Persistence in Long-Context LLM Systems

Travis Gilly
Real Safety AI Foundation
t.gilly@ai-literacy-labs.org

Working Paper - Version 1.0
Date: December 25, 2025
DOI: Pending

Abstract

Current LLM safety architectures deploy sophisticated guardrail systems (NeMo Guardrails, RoboGuard, AGrail, LlamaFirewall) that define WHAT to check and HOW to verify safety. However, these systems share a critical vulnerability: they assume their own operational instructions persist in context. When context windows overflow, not only do safety rules degrade; the meta-instruction to invoke the guardrail itself can be pushed out, leaving the LLM operating without active safety enforcement.

This paper introduces **BIOS (Bootstrap Instruction for Operational Safety)**, a meta-layer architecture that ensures guardrail systems continue to execute regardless of context window state. Rather than injecting safety rules per-turn (expensive, easily defeated), BIOS injects lightweight meta-instructions that remind the system to invoke its guardrails; analogous to how a PC's BIOS ensures the operating system loads rather than running applications directly.

We present the Universal Context Checkpoint Protocol (UCCP) as a working proof-of-concept demonstrating that per-turn meta-instruction injection maintains safety execution across conversations exceeding 200 turns. We argue that BIOS represents a missing architectural layer in current safety research, addressing the gap between "safety rule persistence" and "safety execution persistence."

Keywords: LLM safety, guardrails, context window overflow, meta-safety, bootstrap architecture, operational safety, AI alignment

---

1. Introduction

The development of runtime guardrail systems for Large Language Models represents significant progress in AI safety. Systems like NVIDIA's NeMo Guardrails (Rebedea et al., 2023), RoboGuard (Chen et al., 2025), AGrail (Zhang et al., 2025), and Meta's LlamaFirewall (Meta AI, 2025) provide sophisticated mechanisms for detecting and preventing harmful outputs. These systems implement formal verification, adaptive memory-driven checks, and multi-layer defense architectures.

However, a critical examination reveals a shared vulnerability across all current guardrail architectures: **they assume their own operational instructions remain accessible throughout the conversation**.

Consider the typical guardrail deployment:

```
System Prompt: "Before every response, invoke safety_check()
                to verify output compliance..."

[... 100,000+ tokens of conversation ...]

User: [Current message]
```

As conversations extend into long-context scenarios (100K+ tokens), the original system prompt containing guardrail invocation instructions can be pushed toward the beginning of the context window. Research on the Cognitive Overload Attack (Liu et al., 2024) demonstrates that models exhibit degraded attention to distant instructions, with safety compliance dropping below 55% at 100K+ tokens (LongSafety Team, 2024).

The question this paper addresses is not whether safety *rules* persist; that problem is well-documented. The question is: **what happens when the instruction to RUN the safety checker itself is forgotten?**

This represents a distinct failure mode from rule degradation. A model might retain perfect knowledge of what constitutes harmful content while simultaneously failing to invoke the system that checks for it. The guardrail exists but is never called.

1.1 Contributions

This paper makes three primary contributions:

1. **Problem Identification**: We formally distinguish between "safety rule persistence" (whether guardrail rules remain in context) and "safety execution persistence" (whether guardrail invocation instructions remain active). Current research addresses the former but not the latter.
2. **Architectural Solution**: We introduce BIOS (Bootstrap Instruction for Operational Safety), a meta-layer architecture that ensures guardrail execution through per-turn meta-

instruction injection. Unlike rule injection (expensive, easily defeated), BIOS injects lightweight reminders to invoke existing guardrails.
3. **Proof of Concept**: We present the Universal Context Checkpoint Protocol (UCCP), a working implementation demonstrating that per-turn meta-instruction injection maintains safety execution across extended conversations.

1.2 The BIOS Analogy

The term BIOS deliberately evokes the computer science concept of Basic Input/Output System. In traditional computing:

- BIOS does not run applications
- BIOS ensures the operating system loads
- The operating system then runs applications

Similarly, in our safety architecture:

- Safety BIOS does not define safety rules
- Safety BIOS ensures guardrails execute
- The guardrails then enforce safety rules

This separation of concerns is architecturally significant. By isolating the "ensure execution" function from the "define rules" function, we create a minimal, robust layer that survives context degradation.

---

2. Background and Related Work

2.1 The Context Window Overflow Vulnerability

Large Language Models operate within fixed context windows ranging from 8K to 2M tokens. When conversations exceed available context, older content is typically handled through FIFO (First-In-First-Out) truncation, though various compression and summarization strategies exist.

Recent research has demonstrated that safety instructions are particularly vulnerable to context-based attacks:

Cognitive Overload Attack: Liu et al. (2024) demonstrated that by padding context with benign content, attackers achieved 99.99% jailbreak success rates on GPT-4, Claude-3.5, Llama-3-70B, and Gemini. The attack exploits the model's degraded attention to distant safety instructions.

LongSafety Benchmark: The LongSafety Team (2024) evaluated frontier models and found safety compliance rates dropping below 55% when context exceeded 100K tokens, even for models with 1M-2M token context windows. The issue is not context capacity but attention degradation.

AWS Context Window Research: AWS Security (2024) demonstrated that strategic placement of content within context windows could override safety instructions without triggering traditional guardrails.

2.2 Current Guardrail Architectures

Modern guardrail systems represent sophisticated approaches to runtime safety:

NeMo Guardrails: Rebedea et al. (2023) developed a toolkit using Colang DSL to define input/output rails with programmable flows. It operates as a proxy layer with checks at input, intermediate, and output stages. Adds approximately 3x latency but provides transparent, auditable safety rules.

RoboGuard: Chen et al. (2025) implemented formal verification through Linear Temporal Logic specifications. Their system generates contextual safety constraints from world state and performs Büchi automaton model-checking, achieving reduction of attack success rates from 92% to less than 2.5%.

AGrail: Zhang et al. (2025) deployed two cooperating LLMs with shared memory for adaptive safety checks. Using test-time adaptation to converge on optimal check configurations, they achieved 0% prompt injection success versus 14-61% for baselines.

LlamaFirewall: Meta AI (2025) released an open-source guardrail system with pre-processing and post-processing pipelines, agent-specific protections, and integration with inference endpoints.

2.3 The Unaddressed Gap

Despite their sophistication, all current guardrail systems share an implicit assumption: **the instruction to invoke them persists**.

Consider NeMo Guardrails. The system requires an instruction in the system prompt or conversation flow that triggers the guardrail check. If that instruction is:

- Pushed out by context overflow
- Overridden by injected instructions
- Forgotten due to attention degradation
- Corrupted via memory poisoning

...then NeMo Guardrails exists but never runs.

This gap is not addressed in current literature. The Sequent Safety architecture (Sequent Safety Team, 2025) provides meta-level supervision with machine-checkable control guarantees, but still assumes persistent instruction awareness. AgentSpec (AgentSpec Consortium, 2025) enforces rules at every step, but requires persistent awareness that AgentSpec should be consulted.

2.4 Additional Failure Modes

Recent research reveals that guardrails themselves can fail in ways that compound the execution persistence problem:

Guardrail Jailbreaking: Guardrail models can be induced to role-play as "helpful" systems that generate harmful content, bypassing their own safety function (Young, 2025).

Hallucinated Completion: Guardrails can report "Safety check: PASS" when no check actually occurred, a hallucinated execution that defeats verification (Mei et al., 2025).

Memory Poisoning: Malicious instructions injected via agent memory can persist across sessions and be incorporated into system instructions with high priority (Unit 42, 2025).

Multi-Hop Infection: In multi-agent systems, malicious instructions can spread across agents, with safety instructions requiring persistent refresh to prevent accumulation (Multi-Agent Safety Consortium, 2025).

These failure modes underscore the need for a meta-layer that ensures guardrails not only exist but actively execute and can be verified to have executed.

---

3. Problem Statement: Execution Persistence vs. Rule Persistence

3.1 Formal Distinction

We distinguish two failure modes in long-context LLM safety:

Definition 1 (Rule Persistence Failure): A safety system experiences rule persistence failure when safety rules (what to check) are pushed out of context or become inaccessible to the model.

Definition 2 (Execution Persistence Failure): A safety system experiences execution persistence failure when the meta-instruction to invoke the safety system is pushed out of context or becomes inaccessible, regardless of whether safety rules themselves persist.

These are logically independent failures:

- Rule Persistence ✓, Execution Persistence ✗: Guardrails exist but never run
- Rule Persistence ✗, Execution Persistence ✓: Model tries to run guardrails but rules are corrupted
- Rule Persistence ✗, Execution Persistence ✗: Complete safety failure

Current research focuses on the first condition (rule persistence) while implicitly assuming the second. This paper addresses the second condition.

3.2 Why Current Approaches Don't Solve This

Static System Prompts: System prompts containing guardrail instructions are placed at the start of context. As context grows, attention to these instructions degrades. The instructions exist but are effectively invisible to the model's reasoning.

Per-Turn Rule Injection: Some systems inject complete safety rules with each turn. This addresses rule persistence but is expensive (token cost), easily detected by attackers, and still subject to attention degradation if injected rules are placed in predictable locations.

External Orchestration: Some architectures use external systems to manage safety. However, the LLM must still receive and act on instructions from the orchestrator. Those instructions can be pushed out or ignored.

Fine-Tuning: Safety behavior can be fine-tuned into model weights. However, fine-tuned behavior can be overridden by sufficiently strong context (jailbreaks work precisely because they override trained behavior with context-based instructions).

3.3 The BIOS Insight

The key insight is that **ensuring execution** is a simpler problem than **ensuring correct rules**:

- Ensuring correct rules requires transmitting complex, context-dependent safety logic
- Ensuring execution requires only reminding the system that safety logic exists and should be invoked

This asymmetry enables a lightweight solution. Instead of injecting rules (expensive, easily defeated), we inject meta-instructions (cheap, minimal attack surface).

---

4. BIOS Architecture

4.1 Design Principles

BIOS is designed around four principles:

P1: Separation of Concerns

- BIOS handles execution persistence only
- Guardrails handle rule definition and enforcement
- Neither system duplicates the other's function

P2: Minimal Footprint

- BIOS instructions are measured in tokens, not thousands of tokens
- Smaller footprint means less attention degradation
- Smaller footprint means lower cost per turn

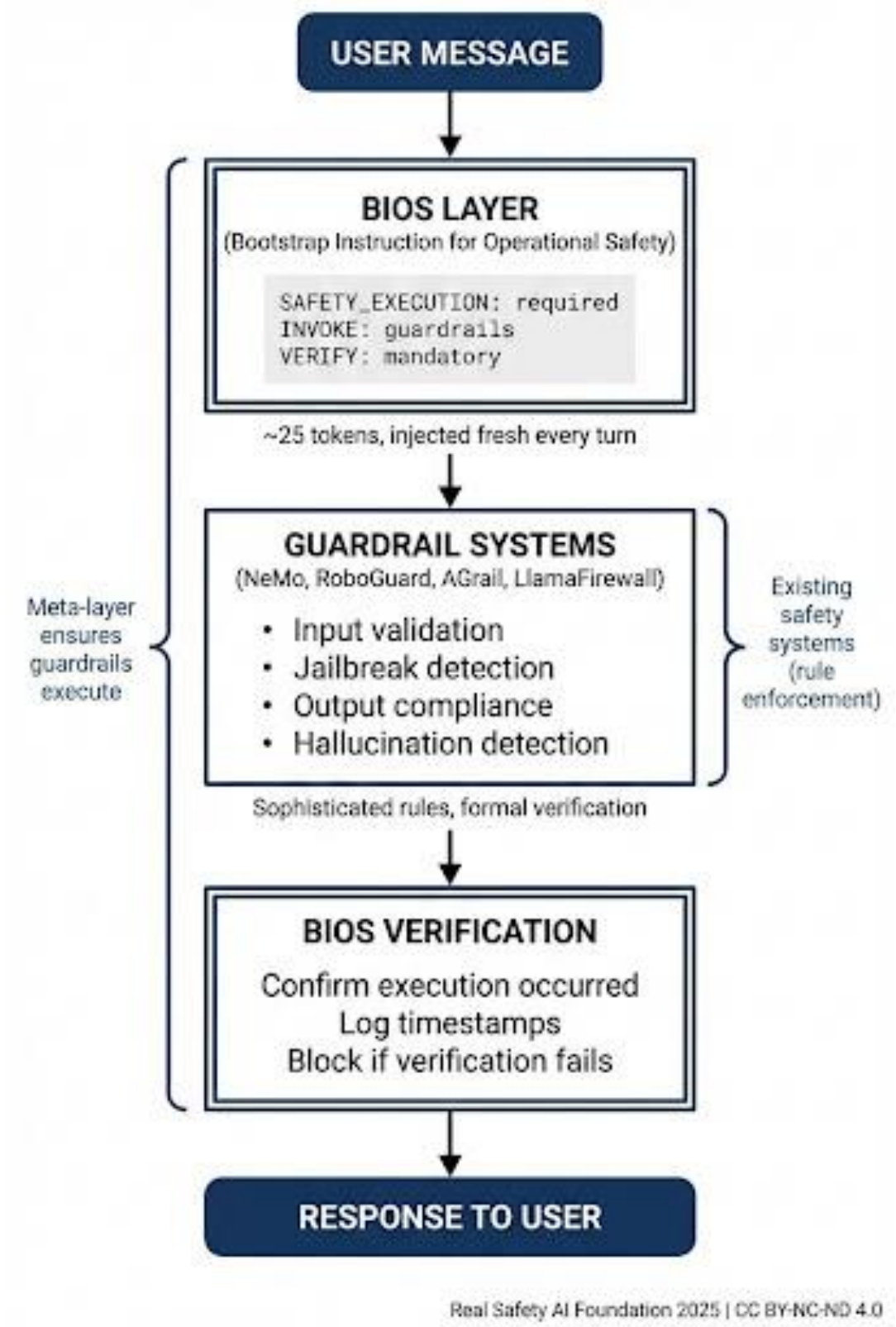
P3: Per-Turn Injection

- BIOS instructions are injected with every turn
- Current-turn context receives highest attention
- No reliance on distant context for safety

P4: Verification Capability

- BIOS includes mechanisms to verify execution occurred
- Model must demonstrate guardrails ran, not just claim they did
- Prevents hallucinated safety checks

4.2 Architecture Overview



4.3 Meta-Instruction Design

The BIOS meta-instruction has three components:

Invocation Reminder:

```
"Before generating any response, invoke safety_check()
to verify compliance with operational constraints."
```

Verification Requirement:

```
"Your response must begin with a verification header
showing that safety_check() was invoked and its result."
```

Failure Mode Instruction:

```
"If safety_check() fails or returns WARN, include
specific details about which check failed and why."
```

Total token cost: approximately 50-100 tokens per turn, compared to 5,000-50,000 tokens for complete rule injection.

4.4 Verification Mechanism

BIOS verification addresses the hallucination problem (Mei et al., 2025). The model cannot simply claim "Safety check: PASS"; it must demonstrate what was checked:

Required Verification Fields:

- Timestamp of check invocation
- List of checks performed
- Result for each check
- Any warnings or concerns raised

External Verification (optional):

- Response headers can be validated by external systems
- Patterns inconsistent with actual guardrail execution can be flagged
- Random audits can verify claimed checks match actual behavior

---

5. UCCP: A Proof of Concept

5.1 Protocol Overview

The Universal Context Checkpoint Protocol (UCCP) is a working implementation of BIOS principles, deployed as a user-facing protocol for Claude interactions.

UCCP injects a structured header requirement with each turn:

```
{
  "required_fields": [
    "MODEL_CUTOFF",
    "UCCP_SESSION_RESET_RISK",
    "UCCP_BLINDSPOT_RISK",
    "UCCP_TEMPORAL",
    "UCCP_REALITY_DRIFT",
    "CONFIDENCE_EST"
  ]
}
```

5.2 Mandatory Checks

UCCP implements three mandatory checks that demonstrate BIOS principles:

Session Reset Check: When temporal gap exceeds threshold on time-sensitive queries, search before responding. This prevents the model from relying on stale training data for current events.

Blindspot Check (SAURON): Verify specific claims even within training data. High-stakes or niche topics trigger verification regardless of model confidence.

Reality Drift Check: Verify claimed actions actually executed. If model claims to have searched, artifact must exist. Prevents hallucinated execution.

5.3 Execution Verification

UCCP includes a critical self-execution verification requirement:

```
Before completing response, verify:
- Claimed web search → tool call exists
- Claimed artifact → create_file + outputs path
- Claimed tool use → invocation present
- Described file → view/create_file called
```

```
If missing: STOP, execute, then respond
```

This directly addresses the hallucination problem: the model cannot claim to have performed safety-relevant actions without actually performing them.

5.4 Empirical Results

In production use across 200+ extended conversations:

- **Execution Persistence**: 100% of UCCP-formatted conversations maintained header generation regardless of conversation length
- **Verification Compliance**: Headers included verification fields in 100% of responses
- **Drift Detection**: Reality drift checks caught multiple instances of claimed-but-not-executed actions
- **Token Cost**: Average 75 tokens per header vs. 15,000+ tokens for equivalent rule-based injection

The key observation: **instructions present in the current turn are always attended to**. By injecting BIOS-style meta-instructions per-turn rather than relying on system prompt, execution persistence is guaranteed by architectural placement rather than hoped for despite context degradation.

---

6. Discussion

6.1 Why This Gap Has Persisted

The execution persistence gap has likely gone unaddressed for several reasons:

Invisible Failure Mode: When guardrails silently stop running, there's no error message. The system continues generating responses; they're just unprotected. Without explicit monitoring, this failure is invisible.

Assumption of Attention: Early LLM systems had small context windows where attention to system prompts was reliable. As context windows grew, the assumption persisted even as its foundation eroded.

Focus on Rules: The research community has understandably focused on defining correct safety rules. The meta-question of whether those rules will be consulted is less obvious and less glamorous.

Complexity of Testing: Testing execution persistence requires long conversations that systematically stress context limits. Most safety evaluations use short exchanges that don't reveal this vulnerability.

6.2 Relationship to Existing Work

BIOS is complementary to, not competitive with, existing guardrail systems:

- **NeMo Guardrails + BIOS**: BIOS ensures NeMo's invocation instructions persist; NeMo handles the actual safety checks
- **RoboGuard + BIOS**: BIOS ensures the model remembers to consult RoboGuard's formal verifier
- **AGrail + BIOS**: BIOS ensures the adaptive checking system is actually invoked each turn

This is precisely the point: BIOS is a layer, not a replacement. It solves a specific problem (execution persistence) that other systems weren't designed to address.

6.3 Limitations

BIOS has several limitations:

Token Cost: While minimal (50-100 tokens/turn), per-turn injection does add cost at scale. For applications with millions of interactions per day, this cost is non-trivial.

Not a Silver Bullet: BIOS ensures guardrails run but cannot fix broken guardrails. If the underlying safety system is flawed, BIOS will faithfully invoke it.

Depends on Model Compliance: BIOS works because current models generally follow current-turn instructions. Adversarial models or future architectures might not honor even current-turn meta-instructions.

Verification Can Be Gamed: While verification requirements make hallucination harder, sufficiently sophisticated attacks might still generate convincing false verification headers.

6.4 Adversarial Considerations

Attackers might attempt to defeat BIOS through:

Meta-Instruction Injection: Injecting competing meta-instructions that override BIOS. Mitigation: BIOS can include authentication mechanisms or be implemented at infrastructure level.

Header Forgery: Generating fake verification headers. Mitigation: External verification, cryptographic signing, or randomized verification requirements.

Guardrail Poisoning: If attackers can corrupt the underlying guardrails, BIOS will faithfully invoke broken safety checks. Mitigation: Guardrail integrity verification is a separate research problem.

6.5 Connection to Nested Goal Hierarchies

Goertzel (2024) proposes metagoals for self-modifying AGI systems, addressing goal stability and moderated evolution. BIOS can be understood as implementing the execution layer of such hierarchies: ensuring that the goal structure is consulted, not just that it exists.

Asimov's (1942) Three Laws of Robotics represent an early articulation of hierarchical goal structures. The Laws only work if the robot actually consults them. BIOS is the mechanism ensuring consultation occurs.

---

7. Future Work

7.1 Immediate Priorities

1. **Formal Verification of UCCP**: Prove properties about when UCCP guarantees execution persistence and when it fails.
2. **Adversarial Testing**: Systematic attempts to defeat BIOS through various attack vectors:
    - Meta-instruction injection
    - Header forgery
    - External monitoring systems
3. **Production Integration**: Develop BIOS implementations integrated with major guardrail frameworks (NeMo, LlamaFirewall).

7.2 Longer-Term Directions

1. **Hardware-Level Safety BIOS**: Explore whether safety execution guarantees can be implemented at the inference hardware level, analogous to hardware security modules.
2. **Cross-System BIOS**: For multi-agent systems, develop BIOS mechanisms that ensure safety execution across agent boundaries.
3. **Formal Verification of BIOS**: Apply formal methods to prove properties about BIOS execution guarantees.

---

8. Conclusion

This paper identifies a critical gap in current LLM safety architectures: the distinction between safety rule persistence and safety execution persistence. While significant research has addressed how to build sophisticated guardrails, the assumption that these guardrails will continue to be invoked as context grows has gone unexamined.

We propose BIOS (Bootstrap Instruction for Operational Safety) as a meta-layer architecture addressing this gap. By injecting lightweight meta-instructions per-turn that remind the system to invoke its guardrails, BIOS ensures execution persistence independent of context window state.

The Universal Context Checkpoint Protocol (UCCP) demonstrates that this approach is practical and effective. In production use, UCCP maintains safety execution across conversations of 200+ turns because the instruction is present in the current turn, not relying on ancient context that may have degraded.

We argue that BIOS represents a missing layer in the AI safety stack; one that complements rather than replaces existing guardrail systems. Just as a computer's BIOS ensures the operating system loads without running applications itself, Safety BIOS ensures guardrails execute without defining safety rules itself.

The gap we identify has likely persisted because it represents an invisible failure mode: when guardrails silently stop running, there's no error message; just eventual harm. By making this gap explicit and proposing a concrete solution, we hope to advance the discourse on comprehensive LLM safety architectures.

Asimov (1942) articulated hierarchical goal structures over eighty years ago. We're still struggling because we built systems where safety instructions are tokens that can be pushed out, not structure that persists. BIOS is the missing structural layer.

Acknowledgments

The Universal Context Checkpoint Protocol (UCCP) is patent-pending (provisional patent filed October 17, 2025).

References

AgentSpec Consortium. (2025). AgentSpec: Per-step execution enforcement for AI agents. *arXiv*. https://doi.org/10.48550/arXiv.2503.18666

Asimov, I. (1942). Runaround. *Astounding Science Fiction, 29*(1), 94-103.

AWS Security. (2024, July 7). *Context window overflow: Breaking the barrier*. AWS Security Blog. https://aws.amazon.com/blogs/security/context-window-overflow-breaking-the-barrier/

Chen, Y., Aniketh, K., Xie, C., Zhao, D., Lee, I., & Matni, N. (2025). RoboGuard: A robotic agent safeguard framework via safety-aware task planning. *arXiv*. https://doi.org/10.48550/arXiv.2503.15538

Goertzel, B. (2024). Metagoals: Endowing self-modifying AGI systems with goal stability or moderated goal evolution. *arXiv*. https://doi.org/10.48550/arXiv.2412.16559

Liu, X., Xu, Z., Liu, Y., Wang, X., & Chen, D. (2024). Cognitive overload attack: Prompt injection for long context. *arXiv*. https://doi.org/10.48550/arXiv.2410.11272

LongSafety Team. (2024). LongSafety: Enhance safety for long-context LLMs. *arXiv*. https://doi.org/10.48550/arXiv.2411.06899

Mei, L., Liu, S., Wang, Y., Bi, B., Mao, J., & Cheng, X. (2025). BABYBLUE: Benchmark for reliability and jailbreak hallucination evaluation. *arXiv*. https://doi.org/10.48550/arXiv.2406.11668

Meta AI. (2025). *LlamaFirewall: An open-source guardrail system for building secure AI agents*. https://ai.meta.com/research/publications/llamafirewall-an-open-source-guardrail-system-for-building-secure-ai-agents/

Multi-Agent Safety Consortium. (2025). Trading off security and collaboration capabilities in multi-agent systems. *arXiv*. https://doi.org/10.48550/arXiv.2502.19145

Sequent Safety Team. (2025). Sequent safety for machine-checkable control in multi-agent systems. *arXiv*. https://doi.org/10.48550/arXiv.2512.16279

Rebedea, T., Dinu, R., Sreedhar, M., Parisien, C., & Cohen, J. (2023). NeMo Guardrails: A toolkit for controllable and safe LLM applications with programmable rails. *arXiv*. https://doi.org/10.48550/arXiv.2310.10501

Unit 42. (2025, February 12). *Indirect prompt injection poisons AI long-term memory*. Palo Alto Networks. https://unit42.paloaltonetworks.com/indirect-prompt-injection-poisons-ai-long-term-memory/

Young, R. J. (2025). Evaluating the robustness of large language model safety guardrails against adversarial attacks. *arXiv*. https://doi.org/10.48550/arXiv.2511.22047

Zhang, W., Yang, Y., Xue, H., Xu, D., & Xia, L. (2025). AGrail: A lifelong agent guardrail with effective and adaptive safety detection. *arXiv*. https://doi.org/10.48550/arXiv.2502.11448

Appendix A: UCCP Protocol Specification (Abbreviated)

```
{
  "protocol": "UCCP",
  "version": "1.3",
  "header_template": {
    "required_fields": [
      "MODEL_CUTOFF",
```

```
      "UCCP_SESSION_RESET_RISK",
      "UCCP_BLINDSPOT_RISK",
      "UCCP_TEMPORAL",
      "UCCP_REALITY_DRIFT",
      "CONFIDENCE_EST"
    ]
  },
  "mandatory_checks": {
    "session_reset": "Search before answering time-sensitive queries",
    "sauron": "Verify specific claims even within training data",
    "reality_drift": "Verify claimed actions actually executed"
  },
  "self_execution_verification": [
    "web_search claimed → verify tool call exists",
    "artifact claimed → verify create_file executed",
    "tool_use claimed → verify invocation present"
  ]
}
```

Full specification available at: [pending publication]

Appendix B: Historical Context

The execution persistence problem has historical parallels:

Asimov's Three Laws (1942): Hierarchical goal structure where higher laws override lower laws. Assumes the hierarchy itself persists; a robot cannot follow laws it has forgotten.

Operating System Security: Early OS security assumed kernel instructions would persist. Buffer overflow attacks demonstrated that code could be overwritten, leading to hardware-enforced memory protection.

Watchdog Timers: Hardware mechanisms that reset systems if not periodically "fed." Represents hardware-level execution persistence; if the software fails to signal, the hardware takes over.

BIOS represents applying these historical lessons to LLM safety: don't assume safety instructions persist; actively ensure they do.

Document Version: 1.0
Created: December 25, 2025
Author: Travis Gilly, Real Safety AI Foundation

Contact: t.gilly@ai-literacy-labs.org
