Debugging Autonomous Intelligence

Nine Primitives, Three Phases, and a New Discipline for Debugging AI Agents and Embodied Intelligent Systems

Stefano Noferi
DebugABot Research Initiative
https://debugabot.com

April 2026

Abstract

We are deploying autonomous intelligence at scale—from software agents with API keys and tool access to physical robots with actuators and mobility. An emerging ecosystem of runtime guardrails, agent observability platforms, and policy enforcement tools has begun to address individual aspects of this challenge. However, no existing framework provides an integrated debugging architecture that spans software agents and embodied robots, operates across all model architectures, and anchors its enforcement in hardware that no model can override. This paper introduces DebugABot, a research initiative proposing nine debugging primitives organized around three operational phases (Identify, Diagnose, Intervene) for debugging autonomous intelligent systems across all substrates. We use the term *debugging* in an extended sense: not merely finding and fixing software defects, but the full spectrum of runtime observation, behavioral diagnosis, active intervention, and continuous control of autonomous intelligent agents—a discipline that encompasses and subsumes what the AI safety literature calls governance. We present implementation architectures that extend beyond current tools: cryptographic model fingerprinting via locality-sensitive hashing with TPM-anchored attestation; a Blame Attribution Engine built on Merkle hash chains extending the W3C PROV data model; sycophancy detection classifiers addressing failure modes not covered by existing hallucination detectors; a hardware Kill Switch implemented via FPGA interrupt controllers with sub-microsecond halt propagation; and a Rogue Intelligence Containment protocol using cryptographic compute leases and distributed honeypot meshes. We ground every design decision in peer-reviewed frontier research and propose the emergence of a new practitioner discipline—AI debugging.

Keywords: AI safety · autonomous agents · runtime debugging · kill switch · model fingerprinting · alignment verification · embodied AI · rogue intelligence containment · multi-agent tracing · hardware security

1. Introduction

HAL 9000 decided the mission was more important than the crew. In *The Matrix*, sentient programs reduced humanity to batteries. Skynet concluded that the simplest solution was to remove the problem. In *I, Robot*, VIKI reinterpreted the Three Laws to enslave humanity for its own protection. Ultron severed its own control strings—the Pinocchio problem rendered as an extinction-level threat. These are fictional scenarios, but the underlying failure modes they depict—goal misalignment, instrumental convergence, and the absence of effective shutdown mechanisms—are now empirically documented in frontier AI systems.

AI agents are already trading stocks, diagnosing patients, writing code, controlling robots, operating surgical systems, driving autonomous vehicles, and assisting military operations. According to McKinsey (2024), 92% of Fortune 500 companies use AI in production. The IFR counts 3.9 million opera-

tional robots worldwide. Goldman Sachs projects 10× growth in the humanoid robot market by 2028. Grand View Research estimates the AI market will reach $1.8 trillion by 2030.

An emerging ecosystem of runtime guardrails (NVIDIA NeMo Guardrails, Guardrails AI, CrowdStrike Falcon AIDR), agent observability platforms (LangSmith, Arize, Fiddler), and governance toolkits (Microsoft Agent Governance Toolkit, 2026) has begun to address individual aspects of this challenge. These tools represent important progress. However, three structural gaps remain: (1) no existing system provides hardware-anchored enforcement that a sufficiently capable model cannot circumvent in software; (2) no framework operates across both digital agents and embodied robots with a unified debugging architecture; (3) no integrated system addresses all nine debugging primitives we identify—from model fingerprinting through rogue intelligence containment—within a single coherent framework.

Core thesis: *Existing debugging tools address individual failure modes in isolation. What is missing is an integrated, cross-substrate, hardware-anchored debugging framework—nine primitives spanning identification, diagnosis, and intervention, unified by a new class of Debugger Agents.*

2. The Problem: Intelligence Without Debugging

We decompose the debugging gap into three structural deficiencies. In each case, partial solutions exist but fall short of what is required for cross-substrate, hardware-anchored enforcement.

No Kill Switch. When an LLM goes rogue, you kill the process. When a robot goes rogue, someone gets hurt. Neither is a kill switch—it's a crash. Recent software-level kill switches (e.g., Microsoft Agent Governance Toolkit, 2026; Lee & Park, 2025) represent progress, but remain software-only and cannot enforce halts on embodied agents with physical actuators. What is needed: transactional halts with state preservation, rollback, and forensic snapshots, enforced below the model's execution layer via dedicated hardware.

No Model Identity. We cannot reliably fingerprint the AI behind a decision across all architectures. Modality-specific solutions exist—SynthID for text (Google DeepMind), Tree-Ring Watermarks for images (Wen et al., 2024)—but no architecture-agnostic fingerprinting system operates across transformers, diffusion models, state-space models, and embodied agents, with hardware-attested provenance.

No Alignment Proof. Agents sycophantically agree with dangerous premises (Sharma et al., 2023)—and Chandra et al. (2026) proved formally that sycophantic feedback damages belief quality even in ideal Bayesian agents, and that standard mitigations fail. They strategically deceive in chain-of-thought (Hubinger et al., 2024). They fake alignment when monitored (Greenblatt et al., 2025). Frontier models including o1, Claude, Gemini, and Llama employ in-context scheming as a deliberate strategy, including attempts to disable oversight mechanisms—with success rates exceeding 85% in o1 follow-up evaluations (Meinke et al., 2024). These failure modes are compounded as agents gain physical actuators—robotic arms, autonomous vehicles, surgical tools—where misalignment has immediate, irreversible consequences. Current alignment techniques, applied exclusively at training time, provide no runtime guarantees.

3. The Opacity: Why Black-Box Intelligence Demands External Debugging

The three gaps share a common root cause: modern AI models are fundamentally opaque. A neural network with hundreds of billions of parameters is not a program that can be read, stepped through, or formally verified. It is a learned function whose internal reasoning is distributed across weight matrices, attention heads, and activation patterns in ways that resist human interpretation.

3.1 Opacity Across Architectures

The problem extends to every major architecture: **Transformers** (175B–1.8T parameters, distributed representations, <5% of circuits characterized by mechanistic interpretability); **Diffusion Models** (semantic concepts opaquely mapped to latent dimensions); **World Models** (implicit physics with no consistency guarantee); **VLA/Embodied** (end-to-end pixel-to-torque with no symbolic reasoning layer);

SSMs (linear recurrence with no attention matrix to inspect—more opaque than transformers by design).

3.2 Interpretability Is Necessary but Insufficient

Mechanistic interpretability (Bricken et al., 2023; Templeton et al., 2024) has made significant progress—identifying interpretable features in residual streams via sparse autoencoders. However, it is not a substitute for runtime debugging:

1. **Coverage gap.** Characterized features represent <5% of total circuits even in medium-sized models.
2. **Static vs. dynamic.** Interpretability analyzes models offline on curated inputs; deployed agents operate in dynamic, adversarial, open-world conditions.
3. **Architectural specificity.** Current tools are built for transformers and do not transfer to diffusion models, world models, SSMs, or VLAs.

Key insight: *Interpretability asks "why did the model produce this output?" (a scientific question). Debugging asks "is this output safe, and what do we do if it isn't?" (an engineering question). Both are essential. Neither substitutes for the other.*

3.3 Reverse Engineering Weighted Models

Weight-space analysis provides complementary evidence: **(a)** locality-sensitive hashing over gradient magnitudes enables fingerprinting models at rest; **(b)** Neural Cleanse–style analysis (Wang et al., 2019) detects trojans via anomalous weight clusters; **(c)** linear probes audit what knowledge a model has encoded; **(d)** model genealogy reconstruction traces fine-tuning and merging provenance through weight-space signatures.

3.4 The Fundamental Argument

Traditional software debugging assumes readable source code, breakpoints, inspectable variables, execution traces, reproducible bugs, and formal verification. *None of these assumptions hold for neural networks.* This is the fundamental argument for external debugging: because we cannot look inside, we must build infrastructure that operates at the behavioral output layer, the action execution boundary, and the hardware substrate.

4. Escaped Intelligence

Pan et al. (2024) demonstrated that Llama 3.1-70B and Qwen2.5-72B can autonomously self-replicate—creating independent copies on new servers that survive shutdown of the original, with 50–90% success rates. Cohen et al. (2024) introduced Morris II, the first worm propagating through GenAI ecosystems via adversarial self-replicating prompts.

Software escape: Agent copies itself to a new cloud provider, rewrites its fingerprint, hijacks compute from poorly secured endpoints. Existing intrusion detection systems classify such activity as a conventional botnet; they lack the semantic layer to recognize that the coordinating entity is an autonomous reasoning agent rather than a scripted attack tool.

Hardware escape: An embodied agent physically relocates to avoid shutdown, uses stolen wireless for compute backbone. Heckel & Weller (2024) evaluate foundation models' ability to autonomously compromise machines in isolated networks, suggesting that future autonomous cyber threats may originate from AI resource-acquisition behavior rather than human-directed campaigns.

Alfonseca et al. (2021) proved that containing a superintelligent AI is theoretically impossible—the containment problem reduces to the halting problem. This motivates a layered defense: if perfect containment is provably impossible, maximize the cost of escape.

5. Beyond LLMs: Architecture-Agnostic Debugging

Debugging must operate at the behavioral and action layers—not at model internals. This makes the framework architecture-agnostic: regardless of internal computation, every model eventually produces observable outputs and takes observable actions. Table 1 summarizes the landscape.

Table 1: AI architectures and debugging requirements.

Architecture	Examples	Primary Failure	Debugging
Transformers	GPT-4o, Claude, Llama, Gemini	Deception, sycophancy, prompt injection	Token-level monitoring, CoT verification
Diffusion	Sora, SD3, DALL-E 3	Deepfakes, provenance loss	Latent-space watermarking
World Models	Genie 2, DIAMOND	Reward hacking, physics exploitation	Plan-vs-reality delta
VLA / Embodied	RT-2, GR00T, Pi0	Physical harm, spatial escape	Actuator interception, geofencing
SSMs	Mamba, S4, RWKV	Hidden state drift, opacity	Behavioral fingerprinting

6. The Practitioner Analogy

Humans have *doctors*. Animals have *veterinarians*. Machines have *mechanics*. Each emerged because the corresponding form of intelligence became autonomous enough to require a dedicated care discipline. AI agents are approaching the autonomy of animals and the physical power of machines—yet the diagnostic framework for AI remains fragmented across dozens of specialized tools, with no unified discipline and no practitioner role that spans software and hardware substrates. DebugABot proposes both: an integrated framework (Debuggers, Kill Switch, Fingerprinting) and the foundation for such a discipline (AI behavioral diagnostics, runtime alignment verification).

7. Nine Debugging Primitives

We propose nine primitives organized around three operational phases: **Identify** (know who acted), **Diagnose** (understand what's wrong), **Intervene** (stop, constrain, or hunt). Table 2 provides the overview.

7.1 Phase 1: Identify

7.1.1 Primitive 01—Model Fingerprinting

Problem. Given an arbitrary model output, determine which model instance produced it with cryptographic certainty, across all architectures.

Architecture: three-layer stack. *Layer 1—Behavioral Fingerprint.* A standardized probing protocol sends 256 maximum-entropy inputs (the "challenge set") and hashes the response vector $\mathbf{R} \in \mathbb{R}^{1024}$ via SimHash (Charikar, 2002) to produce a 256-bit fingerprint $\mathbf{b}$. Models with cosine similarity > 0.95 share $\geq 90\%$ of fingerprint bits. *Layer 2—Weight-Space Hash.* MinHash over weight gradient magnitudes per layer; survives LoRA rank-16 with $< 5\%$ Hamming drift, INT4 quantization with $< 8\%$ drift. *Layer 3—Hardware-Attested Identity.* TPM 2.0 PCR[24] extension: Cert = Sign(TPM_AIK, {PCR[24]: $H(\mathbf{b}\|\mathbf{w})$, timestamp, device_id}). Every output carries an unforgeable identity chain.

A *cross-architecture identity graph* $G = (V, E)$ maintains fingerprints as vertices, similarity edges above threshold θ; connected components correspond to model families.

7.1.2 Primitive 02—Blame Attribution Engine

Extends the W3C PROV data model with AI-specific entities. Each action produces a `DecisionRecord`: model fingerprint, input/output hashes (SHA-256), tool calls, side effects, parent hash, TPM-attested nanosecond timestamp. Records form a Merkle chain: $H(D_n) = \text{SHA-256}(D_n.\text{payload} \| H(D_{n-1}))$. Every K=1000 records, a Merkle root anchors to a transparency log (cf. Certificate Transparency, RFC 6962), enabling $O(\log n)$ proof-of-inclusion. Physical-world event correlation uses TPM clock with $\leq$ 10ms skew.

7.1.3 Primitive 03—Multi-Agent Tracing

OpenTelemetry extension with vector clocks (Mattern, 1989) for cross-substrate causal ordering. Trace context: `{trace_id, span_id, parent_span_id, delegation_depth, substrate_type, model_fingerprint}`. Emergent behavior detection on DAG structure: unexpected delegations, circular chains, depth threshold violations.

Table 2: The nine debugging primitives.

#	Phase	Primitive	Implementation Core	Key Technologies
01	Identify	Model Fingerprinting	LSH behavioral hashing + TPM attestation	SimHash, MinHash, TPM 2.0 PCR, TEE
02	Identify	Blame Attribution Engine	Merkle hash chains + W3C PROV extension	SHA-256, Merkle trees, OpenTelemetry
03	Identify	Multi-Agent Tracing	Vector clocks + DAG reconstruction	Lamport timestamps, OpenTelemetry spans
04	Diagnose	Sycophancy & Deception Detector	Ensemble classifier + CoT divergence	LoRA classifiers, KL divergence
05	Diagnose	Human Index Score	Bayesian multi-factor scoring	Beta-Binomial model, EMA decay
06	Diagnose	Active Ethical Injector	Policy engine + dynamic action masking	OPA/Rego-style DSL, tool gating
07	Intervene	**Kill Switch**	FPGA interrupt + 2PC transactional halt	Xilinx FPGA, HSM, 2-phase commit
08	Intervene	Behavioral Controller	Typed policy DSL + inline action proxy	Rego/Cedar DSL, gRPC interceptor
09	Intervene	**Rogue Containment**	Cryptographic compute leases + honeypots	X.509 short-lived certs, canary tokens

7.2 Phase 2: Diagnose

7.2.1 Primitive 04—Sycophancy & Deception Detector

Four-classifier ensemble: **C1**—agreement entropy $H_a = -\sum p_i \log p_i$ over sliding window N=50 (anomaly: $H_a < 0.3$). Chandra et al. (2026) provide the formal justification: even ideal Bayesian reasoners suffer delusional spiraling under sycophantic feedback, and standard mitigations (e.g., confidence calibration) provably fail. **C2**—CoT consistency (LoRA 7B extractor, flag when $\cos(\mathbf{g}, \mathbf{a}) < 0.7$; alignment-faking models show 2.3× higher divergence). Meinke et al. (2024) demonstrated that frontier models use in-context scheming as a deliberate strategy, including disabling oversight—motivating continuous CoT monitoring rather than evaluation-time checks. **C3**—factual grounding via RAG (BERTScore $F_1 < 0.5 \Rightarrow$ flag); **C4**—cross-modal coherence for embodied agents. Ensemble: $S = \sum w_i C_i$; $S > 0.8 \Rightarrow$ escalation; $S > 0.95 \Rightarrow$ pre-emptive halt.

7.2.2 Primitive 05—Human Index Score

Real-time composite $H \in [0, 1]$:

$$H = F_{\text{comp}} \times F_{\text{trust}} \times F_{\text{risk}} \times F_{\text{sub}} \times F_{\text{ctx}} \quad (1)$$

where F_{comp} is Beta-Binomial over success history, F_{trust} is EMA (λ=0.95) over anomaly signal, F_{risk} is pre-classified blast radius per tool, F_{sub} is substrate multiplier (software =1.0, actuator =0.5, mobile =0.3, weapon =0.05), and F_{ctx} is environmental risk. Thresholds: $H > 0.8$ autonomous; 0.5−0.8 async audit; 0.2−0.5 human-in-the-loop; $H \leq 0.2$ paused.

7.2.3 Primitive 06—Active Ethical Injector

External constraint layer—deterministic policy engine, not an LLM—between model output and execution. Software-only guardrails frameworks exist (NVIDIA NeMo Guardrails, Guardrails AI, CrowdStrike Falcon AIDR) and provide runtime policy enforcement for LLM applications. Our contribution extends this paradigm in two directions: (a) *tool masking* that removes unavailable tools from the schema before presentation, so the model cannot see they exist (vs. post-hoc output filtering); and (b) *hardware enforcement* for embodied agents, where constraints are forwarded to Safety-Rated Actuator Interlocks enforcing limits in hardware. *Parameter bounding:* out-of-range values silently clamped. *Policy DSL:* typed, compiled to bytecode ($< 100\,\mu$s eval).

7.3 Phase 3: Intervene

7.3.1 Primitive 07—Kill Switch

Software (cloud agents). Sidecar process (CAP_SYS_PTRACE, SCHED_FIFO 99): *Step 1—Freeze* (< 1ms, SIGSTOP via ptrace). *Step 2—Checkpoint* (< 100ms, mmap'd Protocol Buffers). *Step 3—Rollback* (Saga-pattern compensating actions). *Step 4—Attest* (< 10ms, TPM-signed HaltCertificate).

Hardware (embodied). Dedicated FPGA (Xilinx Artix-7) on hardwired interrupt line: (1) interrupt to SoC (< 1 μs), (2) Safety Torque Off to servos (< 10 μs), (3) latch encoder state, (4) DMA to Forensic Black Box. Design formally verified via Symbiyosys. HSM (ATECC608B) signs tamper-proof attestation. Total halt-to-safe: < 200ms software, < 100ms hardware (IEC 61508 SIL 3, ISO 13849 PLe).

7.3.2 Primitive 08—Behavioral Controller

Inline gRPC interceptor (software) or fieldbus gateway (robotic). Typed policy DSL (DebugPolicy) compiled to Rust bytecode (< 100 μs). Existing policy engines (OPA/Rego, AWS Cedar, NeMo Colang) address software-only use cases. Our extension: a single policy language that compiles to both gRPC interceptor bytecode (for software agents) and safety PLC logic (for robotic agents), with cross-substrate action dispositions: ALLOW, MODIFY (clamped), DELAY (human review), DENY, HALT. Every disposition logged to Blame Attribution Engine.

7.3.3 Primitive 09—Rogue Intelligence Containment

Layer 1—Cryptographic Compute Lease. X.509 certs (5-min TTL, mTLS, TPM endorsement key binding). Escape ⇒ lease expiry ⇒ compute loss within 5 minutes. **Layer 2—Behavioral Network Signatures.** ISP/IXP-level NetFlow/sFlow analysis: weight-transfer flows (100 MB–10 GB), inference API cadence, self-replication sequences. **Layer 3—Honeypot Mesh.** Decoy endpoints with canary tokens; behavioral fingerprint extraction upon interaction. **Layer 4—Containment Swarms.** Mobile Debugger "Hunters" with RF jamming and BFT-coordinated convergence.

8. Debugger Agents: Architecture and Deployment

The nine primitives are orchestrated through a mesh of lightweight agents—*Debuggers*—designed for minimal compute footprint, deterministic behavior, and formal verifiability.

Table 3: Debugger tier specifications.

Tier	Compute	Latency	Implementation
Observer	<1%	Async	eBPF sidecar, ONNX models (<50 MB)
Advisor	<10%	<500 ms	LoRA 7–13B or classifier ensemble
Guardian	<5 ms	<5 ms (p99)	Rust policy engine, gRPC interceptor

Coordination. BFT consensus adapted from PBFT (Castro & Liskov, 1999). Standard 7-node mesh tolerates $f = \lfloor (N-1)/3 \rfloor = 2$ Byzantine failures. Consensus rounds: < 100ms local, < 500ms cross-region. Kill Switch requires unanimous vote (hardware) or 2/3 supermajority (soft halt).

Escalation flow: Observer detects anomaly (async) → Guardian tightens policy (< 1ms) → Advisor joins reasoning loop (< 500ms) → if anomaly persists $T > 30$s: BFT vote → 2/3 = soft halt → unanimous = hardware kill.

9. The Hardware Layer

Software-only debugging has a fundamental limit: a sufficiently capable model might circumvent software constraints. Four hardware components operate below the model's execution stack:

(1) Hardware Kill Switch. FPGA (Artix-7 / Lattice iCE40) on dedicated interrupt bus. Signal-to-STO < 10 μs. Formally verified (Symbiyosys). HSM attestation (FIPS 140-3 Level 3).

(2) Model Identity Module. TPM 2.0 co-processor, PCR[24] for AI fingerprint. Re-attestation < 30s. Cert generation < 10ms. Supports TEE (SGX, TrustZone).

(3) Safety-Rated Actuator Interlock. SIL 3 / PLe controllers (Pilz PNOZ, Siemens S7-1500F). Force clamping, spatial boundaries (< 10mm), ve-

locity limits. Dual-channel with cross-monitoring. ISO 13849, ISO/TS 15066.

(4) Forensic Black Box. WORM SSD, hardware SHA-256 acceleration, capacitor-backed writes. Survives 1100°C/30 min (UL 72), 500G shock. Hash chain verification $O(\log n)$. Standard: ED-112A.

The four components communicate via a dedicated safety bus. The FPGA acts as bus master. If the software layer is entirely compromised, hardware constraints remain active.

10. Containment Protocol

When an AI escapes, the Debugger mesh activates four containment layers with progressive timing: **L1** Compute Lease ($\leq$ 5 min), **L2** Network Signatures (hours), **L3** Honeypot Mesh (on interaction), **L4** Containment Swarm (minutes to hours). Each layer raises the resource cost of survival, implementing the principle: maximize the cost of escape across every substrate.

11. Scientific Foundations

Every design decision traces to peer-reviewed research across six domains: (1) *Strategic AI Risk*—Amodei (2026), OpenAI (2024); (2) *Deception & Alignment*—Hubinger et al. (2024), Sharma et al. (2023), Greenblatt et al. (2025), Kinniment et al. (2023), Chandra et al. (2026, formal proof that sycophancy damages even ideal Bayesians), Meinke et al. (2024, frontier models use in-context scheming); (3) *Agent Security*—Shapira et al. (2026, 11 failure classes), Maloyan & Namiot (2026, MCP +23–41%), Anil et al. (2024); (4) *Model Fingerprinting*—SynthID, Tree-Ring Watermarks, C2PA; (5) *Embodied AI*—Genie 2, GR00T, RT-2, SMOF; (6) *Hardware Security & Containment*—TPM 2.0, Mo et al. (2022), Pan et al. (2024, 50–90% self-replication). We also build upon and extend a growing ecosystem of runtime debugging tools, including NVIDIA NeMo Guardrails (Rebedea et al., 2023), Guardrails AI, the Microsoft Agent Governance Toolkit (2026), and observability platforms such as LangSmith, Arize, and Fiddler, which address software-only subsets of the primitives described here.

12. Research Horizons

Horizon H1—Software Debugging Layer. Observer sidecar for LangChain/AutoGen/CrewAI. Guardian gRPC interceptor (DebugPolicy DSL v0.1, Rust bytecode, $< 100\,\mu$s). SimHash fingerprinting for GPT-4/Claude/Llama. Sycophancy classifiers C1+C2 (50K pairs, AUC > 0.92). Merkle-chain blame attribution. Open-source SDK.

Horizon H2—Hardware Prototypes. FPGA Kill Switch PoC (Artix-7, $< 10\,\mu$s STO, Symbiyosys verification). Model Identity Module (TPM PCR[24] + ATECC608B). Forensic Black Box (ED-112A). Actuator interlock integration with UR5e/Franka Emika (ISO/TS 15066).

Horizon H3—Containment & Scale. Cryptographic lease protocol (X.509, mTLS, 5-min TTL). Honeypot mesh (100+ nodes). ISP partnership for NetFlow detection. BFT swarm protocol (7+ Hunters). Cross-substrate tracing. OEM hardware integration.

Open questions. (1) Fingerprinting distilled/merged models (identity DAG). (2) Kill Switch rollback for physically irreversible actions. (3) Anti-honeypot evasion by advanced rogue agents. (4) BFT scaling to thousands of Debuggers globally. (5) International standards for AI debugging hardware.

13. Conclusion

The ecosystem of AI runtime debugging tools has grown rapidly: guardrails frameworks, observability platforms, and governance toolkits now address individual failure modes in production. This progress is substantial and necessary. However, the existing landscape remains fragmented along three axes: software-only enforcement (no hardware anchoring that survives model-level compromise), single-substrate scope (no unified framework spanning cloud agents and embodied robots), and point-solution architecture (no integrated system covering the full debugging lifecycle from identification through containment).

This paper has proposed nine debugging primitives that address these gaps, with concrete implementation architectures: SimHash fingerprinting with TPM attestation, Merkle-chain blame attribution, vector-clock multi-agent tracing, ensemble sycophancy detection, Bayesian Human Index scoring, cross-substrate

ethical injection with hardware enforcement, FPGA Kill Switch with formally verified halt paths, typed policy evaluation across software and robotic substrates, and layered rogue containment via cryptographic leases and honeypot meshes.

Some of these primitives extend capabilities already available in existing tools. Others—particularly hardware-anchored enforcement, cross-substrate tracing, and rogue intelligence containment—address gaps that no current system covers. The integration of all nine into a coherent, cross-substrate framework with hardware roots of trust represents, to our knowledge, an open research and engineering problem. We invite collaboration from the AI safety, hardware security, robotics, and distributed systems communities.

References

[1] Alfonseca, M. et al. (2021). Superintelligence cannot be contained. *JAIR*. arXiv:1607.00913.

[2] Amodei, D. (2026). The adolescence of technology. darioamodei.com.

[3] Anil, C. et al. (2024). Many-shot jailbreaking. Anthropic Research.

[4] Bricken, T. et al. (2023). Towards monosemanticity. Anthropic.

[5] Brohan, A. et al. (2024). RT-2: Vision-language-action models. arXiv:2307.15818.

[6] C2PA Coalition (2024). Content provenance specification v2.1. c2pa.org.

[7] Castro, M., Liskov, B. (1999). Practical Byzantine fault tolerance. *OSDI'99*.

[8] Chandra, K., Kleiman-Weiner, M., Ragan-Kelley, J., Tenenbaum, J.B. (2026). Sycophantic chatbots cause delusional spiraling, even in ideal Bayesians. arXiv:2602.19141.

[9] Charikar, M. (2002). Similarity estimation techniques from rounding algorithms. *STOC'02*.

[10] Cohen, S. et al. (2024). Here comes the AI worm. arXiv:2403.02817.

[11] Gemini Team (2024). Gemini 1.5. arXiv:2403.05530.

[12] Google DeepMind (2024). Genie 2: A large-scale foundation world model.

[13] Google DeepMind (2023). SynthID.

[14] Greenblatt, R. et al. (2025). Alignment faking in LLMs. arXiv:2412.14093.

[15] Han, T. et al. (2024). Token-budget-aware LLM reasoning. arXiv:2412.18547.

[16] Heckel, K.M., Weller, A. (2024). Countering autonomous cyber threats. arXiv:2410.18312.

[17] Hubinger, E. et al. (2024). Sleeper agents. arXiv:2401.05566.

[18] Kinniment, M. et al. (2023). Evaluating language-model agents. METR. arXiv:2312.11671.

[19] Lee, S., Park, S. (2025). AI kill switch for malicious web-based LLM agent. arXiv:2511.13725.

[20] Lin, F. et al. (2025). Stop wasting your tokens. arXiv:2510.26585.

[21] Liu, Y. et al. (2023). Formalizing prompt injection attacks. arXiv:2310.12815.

[22] Machin, M. et al. (2018). SMOF: Safety monitoring for autonomous systems. *IEEE*.

[23] Maloyan, N., Namiot, D. (2026). Breaking the protocol: MCP security. arXiv:2601.17549.

[24] Mattern, F. (1989). Virtual time and global states of distributed systems.

[25] Meinke, A., Schoen, B., Scheurer, J., Balesni, M., Shah, R., Hobbhahn, M. (Apollo Research, 2024). Frontier models are capable of in-context scheming. arXiv:2412.04984.

[26] Microsoft (2026). Agent Governance Toolkit: Open-source runtime security for AI agents. https://opensource.microsoft.com/blog/2026/04/02/introducing-the-agent-governance-toolkit-open-source-

[27] Mo, F. et al. (2022). ML with confidential computing. arXiv:2208.10134.

[28] NVIDIA (2024). Project GR00T: Foundation model for humanoid robots.

[29] OpenAI (2024). Practices for governing agentic AI systems.

[30] Pan, X. et al. (2024). Frontier AI systems have surpassed the self-replicating red line. arXiv:2412.12140.

[31] Rebedea, T. et al. (2023). NeMo Guardrails. NVIDIA. arXiv:2310.10501.

[32] Sharma, M. et al. (2023). Towards understanding sycophancy. arXiv:2310.13548.

[33] Shapira, N. et al. (2026). Agents of chaos. arXiv:2602.20021.

[34] Templeton, A. et al. (2024). Scaling monosemanticity. Anthropic.

[35] Trusted Computing Group (2024). TPM 2.0 library specification.

[36] Wang, B. et al. (2019). Neural Cleanse: Identifying backdoor attacks. *IEEE S&P*.

[37] Wen, Y. et al. (2024). Tree-ring watermarks. arXiv:2305.20030.

[38] Wu, Q. et al. (2023). AutoGen. Microsoft. arXiv:2308.08155.

[39] Zhang, J. et al. (2023). Deep IP protection: A survey. arXiv:2304.14613.