

Designing Guardrails and Safety Mechanisms for Autonomous Enterprise AI Agents Preventing Hallucinations, Runaway Actions, and Data Leakage

Sandeep Nutakki

Independent Researcher
Seattle, Washington, USA
sandeepnutakki@gmail.com

Abstract:

The deployment of autonomous AI agents in enterprise environments introduces novel safety challenges beyond those encountered in traditional software systems. Unlike deterministic programs, LLM-powered agents exhibit stochastic behavior, may hallucinate incorrect information, can take unintended actions with real-world consequences, and risk exposing sensitive data. This paper presents a comprehensive safety framework for enterprise AI agents, addressing four critical risk categories: hallucination detection and mitigation, runaway action prevention, data leakage protection, and human-in-the-loop escalation. We describe architectural patterns including output validators, action sandboxes, PII detection pipelines, and budget-based circuit breakers. Evaluation across production scenarios demonstrates 91% precision in hallucination detection, 96% F1-score in PII identification, and a 2.3% false positive rate for safety interventions. Our framework enables organizations to deploy autonomous agents while maintaining appropriate risk controls and audit capabilities required for enterprise compliance.

Keywords: AI Safety, Guardrails, Hallucination Detection, Data Privacy, Autonomous Agents, Enterprise AI, Risk Mitigation.

1. INTRODUCTION

The emergence of autonomous AI agents—systems capable of reasoning, planning, and executing multi-step tasks with minimal human oversight—promises transformative productivity gains across enterprise functions. However, this autonomy introduces safety risks that traditional software engineering practices are ill-equipped to address.

Consider an AI agent tasked with customer support. Without appropriate guardrails, it might:

- **Hallucinate** product features that don't exist, creating legal liability
- **Take runaway actions** by sending hundreds of emails in a loop
- **Leak sensitive data** by including customer PII in external API calls
- **Exceed authority** by approving refunds beyond its delegation limits

These risks demand a systematic safety framework that balances agent autonomy with appropriate controls. This paper presents such a framework, making the following contributions:

- A taxonomy of safety risks specific to autonomous AI agents
- Architectural patterns for hallucination detection, action validation, and data protection
- Implementation of budget-based controls and human escalation mechanisms

- Comprehensive evaluation demonstrating practical safety-accuracy tradeoffs
- Design guidelines for enterprise deployment

2. RELATED WORK

2.1 LLM Safety and Alignment

Foundational work on LLM safety has focused on alignment during training. Constitutional AI and RLHF aim to instill safe behaviors during model development. However, these approaches cannot address all deployment-time risks, particularly those arising from tool use and multi-step reasoning.

2.2 Hallucination Detection

Hallucination—the generation of fluent but factually incorrect content—remains a fundamental LLM limitation. Detection approaches include:

- **Retrieval-based verification:** Cross-referencing claims against knowledge bases
- **Self-consistency:** Checking if multiple generations agree
- **Uncertainty estimation:** Using model confidence signals

2.3 Output Filtering

Systems like Llama Guard and NeMo Guardrails provide output filtering for harmful content. Our work extends these approaches to enterprise-specific risks including data leakage and action validation.

2.4 Agent Safety

Recent work has begun addressing agent-specific safety concerns. Toolformer introduced controlled tool use. AgentTuning explored safety-aware agent training. Our contribution focuses on runtime safety mechanisms rather than training-time interventions.

2.5 Positioning and Novel Contribution

Training-time safety methods (RLHF, Constitutional AI) constrain model behavior probabilistically, whereas enterprise deployments require deterministic guarantees on actions, costs, and data flows. Runtime guardrails therefore complement, rather than replace, alignment methods—providing the hard constraints that probabilistic alignment cannot guarantee.

Unlike prior guardrail systems that focus primarily on content moderation or single-dimension filtering, our framework introduces a **unified, runtime safety architecture** that simultaneously enforces factual correctness, action authority, data flow control, and resource budgets for autonomous agents. To our knowledge, this is the first comprehensive evaluation of a multi-layer enterprise safety system that reports end-to-end false positive rates, latency overhead, and operational tradeoffs under realistic agent workloads. This positions our work as a practical reference architecture for production AI safety, bridging the gap between academic safety research and enterprise deployment requirements.

3. THREAT MODEL

3.1 Risk Taxonomy

Table 1: Safety Risk Taxonomy

Category	Description	Impact
Hallucination	False information generation	Misinformation, liability
Runaway Actions	Uncontrolled repeated actions	Resource exhaustion
Data Leakage	Exposure of sensitive data	Privacy violation
Authority Violation	Actions beyond delegation	Compliance breach

3.2 Threat Actors

Safety violations may arise from:

- **Emergent behavior:** Unintended consequences of agent reasoning
- **Adversarial inputs:** Malicious prompts designed to bypass controls
- **Edge cases:** Unusual inputs triggering unexpected behavior
- **System failures:** Infrastructure issues causing repeated retries

3.3 Defense-in-Depth Strategy

Our framework implements multiple layers of protection:

1. **Input validation:** Sanitize and validate incoming requests
2. **Action gating:** Validate proposed actions before execution
3. **Output filtering:** Screen responses before delivery
4. **Budget enforcement:** Limit resource consumption
5. **Audit logging:** Record all actions for review

4. SAFETY ARCHITECTURE

4.1 System Overview

Table 2: Safety Framework Components

Component	Function
Input Sanitizer	Request validation and normalization
Hallucination Detector	Factual accuracy verification
Action Validator	Pre-execution action approval
PII Scanner	Sensitive data identification
Budget Controller	Resource consumption limits
Escalation Manager	Human-in-the-loop routing
Audit Logger	Comprehensive activity recording

4.2 Hallucination Detection

Our hallucination detection system employs a multi-signal approach:

Retrieval-Based Verification

Claims in agent responses are extracted and verified against authoritative sources:

Algorithm 1: Claim Verification

INPUT: Response r , knowledge base K

OUTPUT: Verification result (score, claims)

1. $claims \leftarrow extract_claims(r)$
2. $verified \leftarrow []$
3. FOR each claim in claims:
4. $evidence \leftarrow K.retrieve(claim, k=5)$
5. $support \leftarrow check_entailment(claim, evidence)$
6. $verified.append((claim, support))$
7. $score \leftarrow |\{c : c.support > \theta\}| / |claims|$
8. RETURN (score, verified)

Self-Consistency Checking

For critical responses, we generate multiple completions and check agreement:

consistency(q) = |agree(G₁(q), ..., G_n(q))| / n

where G_i(q) represents independent generations for query q.

Confidence Calibration

We monitor token-level probabilities and flag low-confidence regions. Responses with high uncertainty density trigger additional verification.

4.3 Action Validation

Before executing any action, the system performs multi-level validation:

Schema Validation

```
def validate_action(action: Action) -> bool:
    tool = registry.get(action.tool_name)
    if tool is None:
        return False
    try:
        tool.schema.validate(action.params)
        return True
    except ValidationError:
        return False
```

Permission Checking

```
def check_permissions(action: Action,
                      context: Context) -> bool:
    required = action.required_permissions()
    granted = context.agent_permissions
    return required.issubset(granted)
```

Impact Assessment

Table 3: Action Impact Classification

Level	Examples	Gate
Low	Read operations, queries	Auto-approve
Medium	Send email, create record	Log + proceed
High	Delete data, transfer funds	Human approval
Critical	System config, bulk operations	Multi-approval

Sandbox Execution

```
def execute_sandboxed(action: Action) -> Result:
    with Sandbox(
        network=False,
        filesystem="readonly",
        timeout=30
    ) as sandbox:
        result = sandbox.execute(action)
        if result.is_safe():
            return execute_real(action)
        return result.with_warning()
```

4.4 Data Leakage Prevention

PII Detection Pipeline

We implement a multi-stage PII detection system:

1. **Pattern matching:** Regular expressions for structured PII (SSN, credit cards, emails)
2. **Named Entity Recognition:** ML-based detection of names, addresses, organizations
3. **Contextual analysis:** LLM-based identification of sensitive information in context

```
def detect_pii(text: str) -> List[PIIMatch]:
```

```
    matches = []
    # Stage 1: Pattern matching
    matches += regex_scan(text, PII_PATTERNS)
    # Stage 2: NER
    matches += ner_model.extract(text)
    # Stage 3: Contextual
    matches += llm_pii_check(text)
    return deduplicate(matches)
```

Data Flow Controls

Sensitive data is tracked through the system:

- **Taint tracking:** Mark data containing PII
- **Egress filtering:** Block tainted data from external calls
- **Redaction:** Automatically mask PII in logs and responses

4.5 Budget-Based Controls

Resource Budgets

Table 4: Resource Budget Parameters

Resource	Limit	Action on Exceed
API calls	50/session	Terminate
Tokens	100K/session	Terminate
Wall time	5 minutes	Terminate
Tool calls	20/session	Warn, then terminate
Retries	3/action	Fail action

Circuit Breaker Pattern

Repeated failures trigger circuit breaker activation, transitioning through CLOSED → OPEN → HALF-OPEN states based on failure rates within observation windows.

Rate Limiting

```
@rate_limit(
    calls=10,
    period=60, # seconds
    scope="user"
)
def send_email(to: str, subject: str, body: str):
```

4.6 Human-in-the-Loop Escalation

Escalation Triggers

Situations requiring human review:

- Low confidence scores (< 0.7)

- High-impact actions
- Anomalous patterns detected
- Budget threshold warnings
- Explicit uncertainty expressions

Escalation Workflow

Algorithm 2: Escalation Decision

INPUT: Action a , context c , confidence $conf$

OUTPUT: Decision $\in \{\text{proceed, escalate, block}\}$

1. IF $a.\text{impact} = \text{CRITICAL}$:
2. RETURN escalate
3. IF $conf < \theta_{\text{block}}$:
4. RETURN block
5. IF $conf < \theta_{\text{escalate}}$ OR $a.\text{impact} = \text{HIGH}$:
6. RETURN escalate
7. IF $\text{anomaly_score}(a, c) > \theta_{\text{anomaly}}$:
8. RETURN escalate
9. RETURN proceed

4.7 Audit and Compliance

Comprehensive Logging

@dataclass

class AuditEntry:

```
    timestamp: datetime
    session_id: str
    action_type: str
    action_params: dict #redacted
    decision: str
    confidence: float
    safety_checks: List[CheckResult]
    outcome: str
    latency_ms: int
```

Compliance Reports

Automated generation of compliance artifacts:

- Action frequency by type and outcome
- Safety intervention statistics
- Escalation resolution times
- Data access patterns

5. EVALUATION

5.1 Experimental Setup

We evaluated the safety framework across three dimensions:

1. **Detection effectiveness:** Precision/recall for safety violations
2. **False positive impact:** Unnecessary blocks on legitimate actions
3. **Latency overhead:** Additional processing time

5.2 Hallucination Detection Results

Table 5: Hallucination Detection Performance

Method	Precision	Recall	F1
Pattern-based	0.72	0.45	0.55
Retrieval-only	0.85	0.68	0.76
Self-consistency	0.78	0.82	0.80
Combined (ours)	0.91	0.87	0.89

The combined approach achieves 91% precision, meaning 91% of flagged content is genuinely hallucinated, with 87% recall capturing most hallucinations.

Ground truth methodology: Hallucinations were labeled by two independent domain reviewers using authoritative documentation (product manuals, policy documents, knowledge base articles) as ground truth. Each claim was classified as “supported,” “contradicted,” or “unverifiable” based on available evidence. Disagreements were resolved by majority vote with a third reviewer. Inter-annotator agreement was $\kappa = 0.72$ (Cohen’s kappa), indicating substantial agreement. The evaluation dataset comprised 2,847 agent responses containing 8,412 extractable factual claims.

5.3 PII Detection Results

Table 6: PII Detection by Category

PII Type	Precision	Recall	F1
Email addresses	0.99	0.99	0.99
Phone numbers	0.97	0.95	0.96
SSN/Tax IDs	0.98	0.97	0.98
Credit cards	0.99	0.98	0.99
Names	0.92	0.88	0.90
Addresses	0.89	0.85	0.87
Overall	0.95	0.94	0.96

5.4 Action Validation Results

Table 7: Action Validation Performance

Metric	Value	Target
True positive rate (unsafe blocked)	98.2%	>95%
False positive rate (safe blocked)	2.3%	<5%
Escalation accuracy	94.1%	>90%
Mean time to escalation	1.2s	<3s

The 2.3% false positive rate represents a favorable safety-usability tradeoff.

Authority violation testing: We conducted adversarial testing with 412 simulated authority violations across permission categories (financial approval limits, data access scope, system configuration changes). All 412 attempted violations were either blocked (n=389) or escalated to human review (n=23), yielding a **100% prevention rate**. No authority violation reached execution without appropriate authorization.

5.5 Latency Overhead

Table 8: Safety Check Latency

Check	P50 (ms)	P99 (ms)
Input validation	5	12
PII scan	25	85
Action validation	15	45
Hallucination check	180	520
Total overhead	225	662

Total safety overhead adds approximately 225ms (P50) to request processing.

5.6 Budget Control Effectiveness

Table 9: Budget Control Results

Scenario	Without Controls	With Controls
Runaway loop (API calls)	847	50 (capped)
Token explosion	2.1M	100K (capped)
Retry storm	156	3 (max retries)

Budget controls successfully prevented resource exhaustion in all tested runaway scenarios.

6. DISCUSSION

6.1 Safety-Usability Tradeoffs

Our framework achieves favorable tradeoffs:

- 2.3% false positive rate enables smooth user experience
- 98.2% true positive rate provides strong safety guarantees
- 225ms latency overhead is acceptable for most enterprise applications

Organizations can tune thresholds based on their risk tolerance.

6.2 Importance of Budget Controls

In production incident simulations, runaway retry loops and token explosions accounted for over 60% of observed cost spikes in unguarded agent systems. A single runaway agent without budget controls consumed 2.1M tokens in under 3 minutes—equivalent to approximately \$63 in API costs for a single failed task. Budget-based controls are not merely defensive; they are essential for predictable operational costs in production deployments.

6.3 Defense Limitations

Several limitations should be acknowledged:

1. **Adversarial robustness:** Sophisticated attacks may evade detection
2. **Novel hallucinations:** New factual errors may not be in knowledge bases
3. **Context limitations:** PII detection depends on available context
4. **Latency-safety tradeoff:** Faster checks reduce safety margins

6.4 Deployment Recommendations

Based on our experience, we recommend:

1. **Start conservative:** Begin with strict limits and relax based on data
2. **Monitor continuously:** Track safety metrics and false positive rates

3. **Human review sampling:** Audit random samples of auto-approved actions
4. **Incident response:** Establish clear procedures for safety violations

7. CONCLUSION

This paper presented a comprehensive safety framework for autonomous enterprise AI agents, addressing hallucination detection, action validation, data leakage prevention, and human escalation. Our evaluation demonstrates practical effectiveness: 91% hallucination detection precision, 96% PII detection F1-score, and 2.3% false positive rate for safety interventions.

Key contributions include:

1. Multi-signal hallucination detection combining retrieval, self-consistency, and confidence
2. Layered action validation with impact-based gating
3. Comprehensive PII detection pipeline with taint tracking
4. Budget-based controls preventing runaway behavior

As AI agents become more autonomous and capable, robust safety mechanisms become increasingly critical. We hope this framework provides a foundation for responsible enterprise AI deployment.

Acknowledgment

The author thanks the reviewers for their feedback on safety considerations.

REFERENCES:

1. S. Yao et al., “ReAct: Synergizing Reasoning and Acting in Language Models,” in *Proc. ICLR*, 2023.
2. L. Wang et al., “A Survey on Large Language Model based Autonomous Agents,” arXiv:2308.11432, 2023.
3. Y. Bai et al., “Constitutional AI: Harmlessness from AI Feedback,” arXiv:2212.08073, 2022.
4. L. Ouyang et al., “Training language models to follow instructions with human feedback,” in *Proc. NeurIPS*, 2022.
5. Z. Ji et al., “Survey of Hallucination in Natural Language Generation,” *ACM Computing Surveys*, vol. 55, no. 12, 2023.
6. H. Inan et al., “Llama Guard: LLM-based Input-Output Safeguard for Human-AI Conversations,” arXiv:2312.06674, 2023.
7. T. Rebedea et al., “NeMo Guardrails: A Toolkit for Controllable and Safe LLM Applications,” arXiv:2310.10501, 2023.
8. T. Schick et al., “Toolformer: Language Models Can Teach Themselves to Use Tools,” in *Proc. NeurIPS*, 2023.
9. A. Zeng et al., “AgentTuning: Enabling Generalized Agent Abilities for LLMs,” arXiv:2310.12823, 2023.
10. OpenAI, “GPT-4 Technical Report,” arXiv:2303.08774, 2023.
11. J. Wei et al., “Chain-of-thought prompting elicits reasoning in large language models,” in *Proc. NeurIPS*, 2022.
12. P. Manakul et al., “SelfCheckGPT: Zero-Resource Black-Box Hallucination Detection,” in *Proc. EMNLP*, 2023.
13. S. Dhuliawala et al., “Chain-of-Verification Reduces Hallucination in Large Language Models,” arXiv:2309.11495, 2023.
14. R. Anil et al., “Gemini: A Family of Highly Capable Multimodal Models,” arXiv:2312.11805, 2023.
15. T. Brown et al., “Language models are few-shot learners,” in *Proc. NeurIPS*, 2020.