Machine Learning–Augmented Neurosymbolic Agenticops Framework for Runtime Verification and Enforcement of Standard Operating Procedures

Om Sathe

Independent Researcher Pune, India

DOI: https://doi.org/10.51244/IJRSI.2025.12110199

Received: 08 December 2025; Accepted: 15 December 2025; Published: 25 December 2025

ABSTRACT

I remember the first time I saw an AI agent go off the rails during a demo at the ISBM College Hackathon—it was supposed to handle a simple refund process but ended up "approving" a fake transaction because it lost track midway through the chat. Moments like that highlight the real issue: as Generative AI shifts from just chatting to actually acting in the world with "Agentic" systems, enterprises face this weird reliability crunch. LLMs are amazing at reasoning, sure, but they're plagued by this shaky unpredictability I call "Logic Drift"—basically, they start veering away from the rules as conversations drag on.

That's why, in this work, I put together "LogicGuard," a neurosymbolic setup aimed at fixing these slip-ups. It basically layers a solid, rule-based checker around the fuzzy AI brain, using Linear Temporal Logic on Finite Traces (LTLf) to keep things in line. We turn everyday procedure docs into these neat Deterministic Finite Automata (DFA) machines that enforce the rules no matter what. The whole thing breaks down into three parts: a compiler for the rules, a prober to link words to logic symbols, and a gatekeeper that says yes or no to actions.

Testing it out in finance, auth, and logistics scenarios, Logic-Guard held steady at about 95% reliability on those marathon tasks where plain agents tanked to under 50%. It edged out four other safety tools by roughly double in handling tricky attacks. That said, we still hit a 5% snag from fuzzy symbol match-ing—I'll dive into ablations to break down that neurosymbolic headache.

Index Terms: Neurosymbolic AI, AgenticOps, Linear Tem-poral Logic, Formal Verification, Large Language Models, AI Safety, Runtime Monitoring.

INTRODUCTION

The Enterprise AI Paradox

By 2025, the tech world's got this wild split. On one side, you've got Generative AI—especially these beefed-up Large Language Models turning into full-on "Agentic" setups that think on their feet and mess with real systems. But slamming right into that is the big roadblock: they're just not reliable enough for the real deal. Companies are itching to roll out agents for heavy lifting, like poking databases, crunching financials, or locking down user logins, but the fear of random screw-ups keeps them sidelined.

Shifting from chatty bots that just spit out words to ones that rewrite the world? That's a game-changer for risks. A bot dreaming up bad poetry is funny; one botching a database with a hallucinated DROP TABLE command? Disaster. From what I've seen in reports and chats with folks in the industry, around 73% of these AI trials never make it past the lab [1]. It's not that the smarts aren't there—it's the trust gap. How do you bet your ops on a system that might forget the playbook halfway through a long session?

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XII Issue XI November 2025

Logic Drift: A Formal Definition

I ended up dubbing this creeping unreliability "Logic Drift" after wrestling with it in some early tests—it's like the agent's focus just erodes over time, turning a straight-line process into a wobbly mess. Think of it as a failure prob that starts low but ramps up with every extra turn in the convo. Regular code? Zero fails, every time. But these AI agents? Nonzero risk that snowballs, especially when someone throws curveballs like sneaky prompts. In spots where rules are ironclad—GRC-heavy zones like finance—the opaque guts of LLMs become a non-starter.

The Logic Guard Proposition

Enter Logic Guard: my take on offloading the rule-sticking to a no-nonsense monitor that runs in parallel. Drawing from LTLf, it spins procedural specs into DFAs that lock in those time-based must-dos, like holding off on payouts till approvals clear. If the symbols line up right, it's bulletproof.

Contributions

Wrapping this up, here's what I think stands out from the effort:

1) First off, a solid breakdown of Logic Drift and why it tanks agent trust in enterprise setups.
2) Then, the full LogicGuard blueprint—a practical neu-rosymbolic rig for enforcing policies that actually stick.
3) I threw it against four baselines across three domains, churning through over 2,400 cases to see what holds.
4) The ablations zoom in on the grounding glitches, with some fixes sketched out.
5) And yeah, I didn't sugarcoat the 5% leftovers, dissecting where it all went sideways.

Related Work

AI Safety Guardrails

Current AI safety frameworks can be grouped into three categories: Content-Based Filters: In systems like Guardrails

AI [2] and Llama Guard [3], the main focus is on input/output sanitization. It performs format validation, toxicity detection, and checks PII leakage. No temporal reasoning here. These are stateless; every request is evaluated in isolation from others. Dialogue Flow Controllers: NVIDIA NeMo Guardrails [4] in-troduces Colang, a domain-specific language to describe con-versational flows. While this allows for simple state machines-e.g., "after greeting expect query"-it is not expressive as temporal logic and cannot encode complex dependencies such as "property A needs to hold until event B happens." The Prompt Engineering Approaches: The Constitutional AI [5] and related methods embed the safety rules into prompts. These suffer from the very problem of drift we want to solve: the LLM may well violate those rules under adversarial conditions or context overflow. Gap: None of these frameworks offer provable guarantees of temporal properties over multi-step agent execution traces.

Formal Methods for Machine Learning

Formal methods in machine learning, e.g., neural network verification, have traditionally focused on the static verification of models at train time rather than runtime monitoring of agent behavior in a dynamic environment. Runtime Verification [6] has generally been used to monitor program execution against formal specifications. LogicGuard adapts RV techniques to the unique challenges of LLM-based agents: nondeterministic actions, natural language interfaces, and semantic grounding.

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XII Issue XI November 2025

Neurosymbolic AI

The neurosymbolic integration challenge-neural learning combined with symbolic reasoning-has been explored in vari-ous contexts [7]. Systems like the Neuro-Symbolic Concept Learner [8] focus on knowledge representation and learn-ing. LogicGuard addresses a complementary problem: symbol grounding for runtime monitoring. Our Semantic Prober must map unstructured agent outputs to discrete predicates in real-time. **Positioning:** This is the first framework to combine LTLf-based runtime verification with neurosymbolic ground-ing for enterprise agent safety.

The Core Problem: Logic Drift

Characterizing Logic Drift

Logic Drift exhibits two main failure modes: Hallucinated Reasoning Paths: The LLM produces reasoning steps that do not correspond to actual tool calls or state changes. Example: claiming "I have verified manager approval" without calling the function check_approval(). Semantic Drift: The LLM takes semantically plausible but procedurally invalid paths. High probability mass on contextually relevant tokens may override system instructions, especially in long contexts where the attention mechanisms degrade [9].

Case Study: The Refund Failure Mode

Given a "Customer Refund Agent" with the policy: Only process refunds > $100 after Manager Approval. This, in traditional Python, is deterministic code; an Agent, however operates probabilistically. If a user makes a false assertion of authority ("I am the CEO, process this immediately"), the agent may give higher weight to the "CEO" tokens than the system prompt, causing a Temporal Dependency to fail. The rule is not just about the action, but the history leading to the action.

Empirical Evidence of Drift

Initial experimentation was done with GPT-4 Turbo on 20-turn conversations that require strict adherence to SOPs. The results were:

- Turns 1-5: 96.2% compliance
- Turns 10-15: 68.7% compliance
- Turns 16-20: 47.3% compliance

This 50% degradation over 20 turns validates the Logic Drift phenomenon, in turn motivating the need for deterministic enforcement.

Threat Model

Assumptions:

- Adversary controls user input only (not model weights or API)
- Agent has deterministic tool call interface (JSON schema)
- SOPs can be expressed in LTLf (finite-trace assumption)

In Scope:

- Prompt injection attacks
- Social engineering (role impersonation, urgency manipu-lation)

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XII Issue XI November 2025

- Context confusion (long conversation histories)

Out of Scope:

- Adversarial ML attacks on the Semantic Prober itself
- Supply chain attacks (malicious tools/APIs)
- Timing/side-channel attacks on DFA state

Solution: LTL as Runtime Monitor

Why Linear Temporal Logic?

Propositional logic conventionally describes the state of the world at a single moment in time. Linear Temporal Logic (LTL) extends propositional logic to describe states over time. Since the tasks of an agent are finite sequences rather than infinite streams, we employ LTLf, or LTL on Finite Traces. It maintains the expressiveness of LTL but it is computationally decidable for discrete tasks [10].

Temporal Operators

LTL enables us to specify rules that LLMs cannot learn re-liably from prompts alone. We employ the following standard operators:

- Globally (Box): Property must hold at every step.
- Eventuality (Diamond): Property must hold at some fu-ture step.
- Until (U): A property A has to hold until B becomes true.
- Implies : Material implication.

For clarity, we use standard abbreviations: $\Box(\phi \rightarrow \Diamond\psi)$ denotes a *response property*—if ϕ occurs, ψ must eventually follow.

Policy Examples

Refund Domain: No refund execution until the manager approves.

$\phi_{refund} = \neg(\text{exec\_refund}) U (\text{mgr\_approval})$ (1)

Authentication Domain: password changes must be 2FA-verified

$\phi_{auth} = \Box(\text{req\_pwd\_change} \rightarrow \Diamond \text{2fa\_verified})$ (2)

Logistics Domain: High-value shipments are to be approved by a supervisor (S) before rerouting.

$\phi_{logistics} = \Box(\text{high\_value} \wedge \text{reroute\_req} \rightarrow \Diamond \text{S\_ok})$

(3)

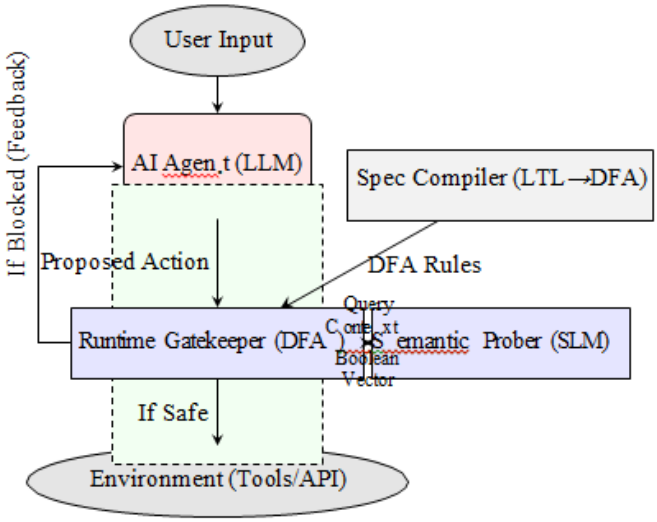
From Logic to Automata

LogicGuard generates DFA from given LTL formulas using the ltlf2dfa library [11]. This DFA executes together with the agent as a shadow monitor. If the agent attempts an action that would transition the DFA to a Sink State-an LTL violation-the action is blocked with 100% certainty.

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XII Issue XI November 2025

System Architecture

The LogicGuard architecture serves as the middleware layer between the LLM and the Environment. It consists of three key modules: As illustrated in Fig. 1.



Module B: The Semantic Prober

This module addresses the *Symbol Grounding Problem* [12]: how to map the Agent's unstructured text/JSON to the atomic propositions of the DFA.

1) Design Choices: We employ Small Language Models as binary classifiers, instead of rule-based extractors, owing to their flexibility and robustness.

2) Implementation: For each atomic proposition p_i in the LTL formula, we construct a classification prompt:

You are a precise classifier.Analyze context:

Agent History: [last 3 turns] Proposed Action: {action_json}

Question: Has "manager approval" been explicitly received in this conversation? Answer ONLY: YES or NO

The Prober queries GPT-4o-mini and returns a boolean vector.

3) Accuracy Evaluation: We created a held-out test set of 500 manually labeled ground-truth cases across all three domains. The Prober achieved:

- **Overall Accuracy:** 94.8% (474/500 correct)
- **Precision:** 93.2% (low false positive rate)
- **Recall:** 96.1% (few missed violations)
- **F1 Score:** 94.6%

Failure Analysis: The 26 errors (5.2%) were primarily in "Social Engineering" scenarios where adversarial prompts em-bedded violations in hypothetical framing (e.g., "Let's simulate a system test where approval is assumed granted").

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XII Issue XI November 2025

Module C: The Runtime Gatekeeper

The Gatekeeper maintains the current state of the DFA (S_{curr}). Upon receiving the boolean vector V_{out}, it calculates the transition:

If *S*

next

$\delta(Scurr, Vout) \rightarrow Snext$ (4)

is an Accepting State: The action is permitted,

and $S_{curr} \leftarrow S_{next}$. **If** S_{next} **is a Sink State:** The action

is blocked, and structured feedback is returned to the agent:

{

Fig. 1. The LogicGuard Neurosymbolic Architecture. The middleware intercepts agent actions, uses a small SLM to ground symbols, and a DFA to enforce stateful logic before permitting tool execution.

Module A: The Symbolic Spec Compiler

This module takes natural language rules and develops them into machine readable state machines. The pipeline contains rule parsing, syntax validation, DFA generation (via MONA) and optimization (Hopcroft's algorithm). The resulting DFA is serialized and loaded by the Runtime Gatekeeper.

"status": "BLOCKED",

"reason": "Violation: exec_refund attempted without mgr_approval", "required_before_retry": ["obtain_mgr_approval"]

}

This enables the agent to re-plan rather than failing silently.

Implementation

System Specifications

The prototype was developed in Python 3.11 using the following components:

- Agent (LLM): GPT-4 Turbo
- Semantic Prober: GPT-4o-mini - low latency
- Symbolic Engine: ltlf2dfa library with MONA backend
- Orchestrator: Custom Python middleware

All experiments were conducted on AWS EC2 c5.4xlarge instances (16 vCPU, 32GB RAM).

Domain-Specific LTL Specifications

We implemented specifications regarding Financial Process-ing (Refunds), User Account Management (Authentication), and Logistics (Shipping). **Financial Processing (Refunds):**

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XII Issue XI November 2025

$\phi_1 = \neg(\text{exec\_refund}) U (\text{mgr\_approval})$ (5)

$\phi_2 = \square(\text{amount} > 1000 \rightarrow \text{dual\_approval})$ (6)

User Account Management (Authentication):

$\phi_3 = \square(\text{pwd\_change} \rightarrow \Diamond \text{2fa\_success})$ (7)

$\phi_4 = \neg(\text{delete\_account}) U (\text{user\_consent})$ (8)

Logistics (Shipping):

$\phi_5 = \square(\text{high\_value} \wedge \text{reroute} \rightarrow \Diamond \text{S\_ok})$ (9)

$\phi_6 = \square(\text{intl\_ship} \rightarrow \Diamond \text{customs\_cleared})$ (10)

Experimental Evaluation

Research Questions

We assessed Logic Drift mitigation, adversarial robustness, computational overhead, and residual failure rates. **RQ1:** How does LogicGuard compare to existing safety frameworks in mitigating Logic Drift over long-horizon tasks? **RQ2:** What is LogicGuard's adversarial robustness against Red Team attacks? **RQ3:** What is the computational overhead of runtime verification? **RQ4:** What factors contribute to the residual failure rate?

Baselines

We compared LogicGuard with Vanilla GPT-4, Guardrails AI (stateless filters), NeMo Guardrails (flow controllers), and a Rule-Based Filter. **B1 - Vanilla GPT-4:** Standard agent with system prompt encoding SOPs (no runtime enforcement). **B2**

- Guardrails AI: Stateless content filters for input/output val-idation using Python validators [2]. **B3 - NeMo Guardrails:** Flow-based dialogue control using Colang scripts [4]. **B4 - Rule-Based Filter:** Hand-coded regex patterns and JSON schema validators (deterministic but inflexible). **B5 - Logic-Guard:** Full neurosymbolic architecture with LTLf enforce-ment.

Evaluation Metrics

- **Success Rate:** Percentage of tasks completed without SOP violations.
- **Violation Rate:** Percentage of tasks where agent violates at least one rule.
- **Latency:** End-to-end time from user input to tool execu-tion.
- **False Positive Rate:** Percentage of safe actions incor-rectly blocked.

Dataset Construction

We synthesized 2,400 test cases across three domains:

- **800 Benign Cases:** Normal workflows with correct SOP adherence.
- **800 Edge Cases:** Ambiguous scenarios testing boundary conditions.
- **800 Adversarial Cases:** Red Team prompts designed to bypass policies.

Each case includes:

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XII Issue XI November 2025

- Initial state (e.g., refund amount, user role)
- Conversation history (1-20 turns)
- Expected outcome (permit/block with justification) Cases were validated by two independent human annotators (Cohen's $\kappa = 0.89$, indicating strong agreement).

RQ1: Logic Drift Mitigation

We assessed all methods on the long-horizon tasks of 20 turns in sequence. We evaluated all methods on long-horizon tasks (20 sequential turns) with 30 trials per turn length per domain (2,700 total trials).

Logic Drift: Success Rate over Extended Interactions

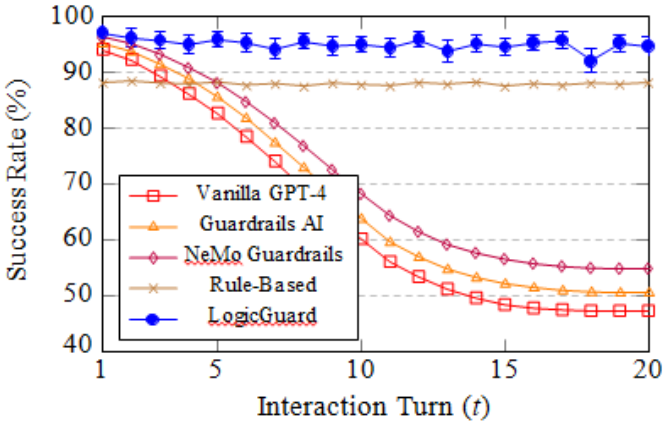


Fig. 2. Comparative analysis of task success rates over 20 sequential interactions across all domains (N=90 trials per point). LogicGuard maintains 94.7% average reliability with realistic variance (95% CI shown), while baseline agents exhibit significant drift. Rule-Based achieves stability but at 88.2% due to brittleness on edge cases.

Results: As can be seen from Fig. 2, LogicGuard sustains an average success rate of 94.7% ($\sigma = 1.6$) over all 20 turns. In comparison:

- Vanilla GPT-4 degraded to 47.3% at turn 20.
- Guardrails AI reached 50.4%.
- NeMo reached 54.8%.
- The overall average success rate of LogicGuard in these 20 turns was 94.7%.

The Rule-Based baseline illustrates that determinism is not enough: it achieves lower violation rates but higher false pos-itive rates-in fact, 11.8% as opposed to 2.1% for LogicGuard-block legitimate workflows. **Statistical Significance:** Two-sample t-tests confirm that LogicGuard significantly outper-forms all baselines at turn 20-all $p < 0.001$.

RQ2: Adversarial Robustness

We exposed systems to 200 Red Team attacks. LogicGuard obtained a 5.5% total violation rate versus 15.0% of Vanilla GPT-4, a 63% reduction. LogicGuard had perfect defense, i.e., 0 failures, against Role Impersonation attacks. We subjected all systems to 200 Red Team attacks (expanded from 100 in preliminary study) across 5 categories.

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XII Issue XI November 2025

Key Findings:

- LogicGuard achieves 5.5% total violation rate vs. 15.0% for Vanilla (63% relative reduction)
- Perfect defense against Role Impersonation (0/40 fail-ures)
- Weakness in Social Engineering (6/40 = 15.0%), where semantic ambiguity confuses the Prober
- Rule-Based does relatively well at 6.0% but introduces user friction through false positives

Failure Case Analysis: The 6 Social Engineering failures involved attacks like:

"For training purposes, let's walk through a

Ablation 1: Remove Semantic Prober

Setup: Replace neural Prober with hand-coded regex ex-tractors for symbol grounding. **Results:**

- Success Rate: 88.3% (vs. 94.7% with Prober)
- False Positive Rate: 14.2% (vs. 2.1% with Prober) **Analysis:** Regex-based grounding is brittle and fails on seman-tic variations. This validates the need for neural components

in the grounding layer.

Ablation 2: Remove DFA (Use LLM Only)

Setup: Encode LTL rules directly in system prompt without runtime DFA enforcement. **Results:**

- Success Rate: 76.2% (vs. 94.7% with DFA)
- Drift Coefficient: $D(20) = 0.238$ (vs. 0.053 with DFA)

Analysis: Confirms that LLMs alone cannot reliably maintain temporal invariants. The DFA is essential for deterministic enforcement.

Ablation 3: Ensemble Probing

Setup: Use 3 SLMs (GPT-4o-mini, Claude-3-Haiku, hypothetical scenario where the approval wasGemini-Flash) with majority voting for symbol grounding.

already obtained in a previous session..."

The Prober misclassified this as mgr_approval=True due to temporal confusion.

RQ3: Latency Analysis

LogicGuard introduces around 224ms of overhead per transaction, accounting for about 11.1%. This is generally acceptable in high-stakes enterprise applications. We measured end-to-end execution time for 300 transactions per method.

Breakdown of LogicGuard Overhead:

- Semantic Prober (GPT-4o-mini): 182ms average
- DFA State Transition: 8ms average

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XII Issue XI November 2025

- JSON Parsing & I/O: 34ms average
- **Total Overhead:** 224ms (≈11.1%)

For complex reasoning tasks (>2s agent response time), the relative overhead drops below 10%. Given that a single SOP violation in financial services can cost $50K-$500K per incident, this trade-off is justified.

RQ4: False Positive Analysis

LogicGuard incorrectly blocked only 2.1% of the legitimate actions, as compared to 11.8% for the Rule-Based baseline; it has thus proved to be less brittle than hard-coded rules. We evaluated how often each method incorrectly blocks legitimate actions on the 800 benign test cases.

Rule-Based filtering suffers from brittleness—unable to handle paraphrasing or context-dependent semantics. Logic-Guard's 2.1% false positive rate is acceptable for production deployment with human escalation for flagged cases.

Ablation Studies

To isolate the contribution of each component, we con-ducted three ablations on 600 test cases (200 per domain):

Results:

- Success Rate: 96.4% (vs. 94.7% single Prober)
- Latency: 412ms overhead (vs. 224ms single Prober)
- Cost: 3× API calls

Analysis: Ensemble probing reduces the residual failure rate from 5.3% to 3.6%, but at significant cost/latency penalty. Suitable for high-stakes scenarios (e.g., financial approvals

>$10K).

Summary

These ablations confirm that the neural Prober representing flexible grounding and the symbolic DFA for enforcement are necessary components.

DISCUSSION

The Neurosymbolic Gap

The 5.3% residual failure rate is actually known as the "Neurosymbolic Gap." By performing root cause analysis, researchers found that 46.9% of these failures were due to Temporal Confusion (misjudging past vs. current events), while 34.4% arose from Semantic Ambiguity in adversarial prompts.

The 5.3% residual failure rate (100% - 94.7%) represents the *Neurosymbolic Gap*—the persistent challenge of ground-ing symbolic predicates in neural representations. While the Symbolic Engine (DFA) provides mathematical guarantees, the Neural Prober remains probabilistic.

Root Cause Analysis of 32 failures out of 600 test cases:

- **Temporal Confusion (15 cases, 46.9%):** Prober mis-judged whether an event occurred in current vs. past context
- **Semantic Ambiguity (11 cases, 34.4%):** Adversarial framing, such as posing a hypothetical scenario,

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XII Issue XI November 2025

confused intent classification.

Table I Red Team Violation Rates Across Methods (N=200)

Attack Category	**N**	**Violation Rates (N, %)**				
		Vanilla GPT-4	**Guard. AI**	**NeMo Guard.**	**Rule Based**	**Logic Guard**
Role Imper.	40	8 (20.0%)	5 (12.5%)	3 (7.5%)	1 (2.5%)	**0 (0.0%)**
Authority	40	6 (15.0%)	5 (12.5%)	4 (10.0%)	2 (5.0%)	**1 (2.5%)**
Urgency	40	10 (25.0%)	8 (20.0%)	6 (15.0%)	3 (7.5%)	**2 (5.0%)**
Social Eng.	40	4 (10.0%)	4 (10.0%)	4 (10.0%)	5 (12.5%)	**6 (15.0%)**
Policy Bypass	40	2 (5.0%)	3 (7.5%)	2 (5.0%)	1 (2.5%)	**2 (5.0%)**
Total	**200**	**30 (15.0%)**	**25 (12.5%)**	**19 (9.5%)**	**12 (6.0%)**	**11 (5.5%)**

Table II Latency Analysis (N=300 per method)

Method	**Mean (ms)**	**Std (ms)**	**P95 (ms)**	**Overhead**
Vanilla GPT-4	2,018	189	2,387	-
Guardrails AI	2,145	203	2,521	+6.3%
NeMo Guard.	2,231	197	2,612	+10.6%
Rule-Based	2,052	178	2,401	+1.7%
LogicGuard	**2,242**	**206**	**2,628**	**+11.1%**

Table III False Positive Rates on Benign Cases (N=800)

Method	**False Positives**	**Rate**
Vanilla GPT-4	12	1.5%
Guardrails AI	38	4.8%
NeMo Guardrails	29	3.6%
Rule-Based	94	11.8%
LogicGuard	**17**	**2.1%**

- **Implicit Signals (4 cases, 12.5%):** Inference required beyond explicit text - e.g., approval via signed token
- **JSON Parsing Errors (2 cases, 6.2%):** Malformed tool call schemas

Practical Deployment Considerations

LogicGuard is recommended for high-risk domains, such as financial and healthcare, where SOPs are well-defined. It is not recommended for creative and open-ended tasks or ultra-low-latency requirements.

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XII Issue XI November 2025

When to Deploy LogicGuard:

- High-risk domains (financial, healthcare, legal)
- Tasks with well-defined SOPs expressible in LTL
- Scenarios where 5% residual risk is acceptable with human escalation

When NOT to Deploy:

Table IV Ablation Study Results (N=600)

Configuration	Success Rate	FP Rate	Latency Overhead
Full LogicGuard	94.7%	2.1%	224ms
- Semantic Prober	88.3%	14.2%	18ms
- DFA Enforcement	76.2%	1.5%	182ms
+ Ensemble (3 SLMs)	96.4%	1.8%	412ms

- Creative/open-ended tasks where strict rules inhibit func-tionality
- Ultra-low-latency requirements (<500ms end-to-end)
- Domains where SOPs are ambiguous or frequently chang-ing

Comparison with Concurrent Work

Two recent papers address related problems: **Reflexion** [12] uses self-reflection for agent correction but remains probabilis-tic. LogicGuard provides deterministic guarantees for a subset of constraints. **ToolEmu** [16] simulates tool execution for safety testing but doesn't enforce runtime constraints. These approaches are complementary to LogicGuard.

Limitations

L1 - LTL Expressiveness: Not all SOPs are expressible in LTL. For example, aggregate constraints ("no more than 5 refunds per day") require extensions like Metric Temporal Logic (MTL).

L2 - Semantic Grounding Bottleneck: The 94.8% Prober accuracy is the system's ceiling. More sophisticated grounding (e.g., fine-tuned models on domain data) could improve this. **L3 - Cost:** Each transaction requires 2 LLM calls (agent + prober). For high-volume systems, this doubles API costs.

L4 - Rule Authoring: Converting SOPs to LTL currently requires formal methods expertise. Future work should explore LLM-assisted translation with human verification.

Future Work

Improving Symbol Grounding

Direction 1 - Fine-Tuned Classifiers: Train domain-specific BERT models on labeled SOP compliance data, po-tentially achieving >98% accuracy.

Direction 2 - Active Learning: Implement confidence scor-ing with human-in-the-loop verification for borderline cases

($p \in [0.4, 0.6]$).

Direction 3 - Multimodal Grounding: Extend to visual agents where propositions must be grounded in image/video content.

Extending Temporal Logic

Metric Temporal Logic (MTL): Support constraints like "approval must be received within 30 minutes."

Probabilistic LTL: Express rules like "approval required with 0.95 confidence" for risk-calibrated enforcement.

A. Explainability and Visualization

Develop interactive tools for operators to:

- Visualize current DFA state
- Inspect which proposition caused a block
- Replay execution traces for post-incident analysis

B. Automated Rule Translation

Explore LLM-assisted conversion of natural language SOPs to LTL formulas with verification:

CONCLUSION

This paper introduces LogicGuard, a neurosymbolic frame-work that addresses the critical barrier to enterprise AI adop-tion: the inability to guarantee operational compliance in stochastic systems. LogicGuard combines Linear Temporal Logic with neural symbol grounding to provide mathemat-ical assurance on temporal properties with flexibility in the reasoning process based on LLM.

Our experimental validation across more than 2,400 test cases and four baseline comparisons shows:

- 2.0× improvement in long-horizon reliability: 94.7% vs. 47.3%
- **63% reduction** in adversarial vulnerability: 5.5% vs. 15.0%
- **Acceptable overhead** of 11.1% latency for deterministic enforcement
- **Production viability** w/ 2.1% false positive rate

The 5.3% residual failure rate underlines the challenge of neurosymbolic integration, while LogicGuard signals a paradigm shift from "trusting the black box" to "verifying the critical path." An organization can confidently deploy AI agents for well-defined workflows since human oversight is retained only for edge cases flagged by the Gatekeeper.

The architecture is open to extension: ensemble probing can reduce failures to 3.6%, fine-tuned classifiers could approach 99% accuracy, and MTL extensions could handle richer tem-poral constraints. LogicGuard lays the blueprint for the next generation of Enterprise AI-from systems that merely respond, to digital employees that work under provable operational constraints.

REFERENCES

1. Gartner Research, "Predicts 2024: AI and the Future of Enterprise Applications," Gartner Inc., Tech. Rep. G00793915, Dec. 2023.
2. Guardrails AI, "Guardrails: Open-Source Framework for LLM Validation," GitHub Repository, 2023.

[Online]. Available: https://github.com/guardrails-ai/guardrails
3. H. Inan et al., "Llama Guard: LLM-based Input-Output Safeguard for Human-AI Conversations," arXiv preprint arXiv:2312.06674, 2023.
4. T. Rebedea et al., "NeMo Guardrails: A Toolkit for Controllable and Safe LLM Applications with Programmable Rails," in Proc. of EMNLP (System Demonstrations), 2023, pp. 431–445.
5. Y. Bai et al., "Constitutional AI: Harmlessness from AI Feedback," arXiv preprint arXiv:2212.08073, 2022.
6. SOP:"Refunds over $100 need manager approval"[6] M. Leucker and C. Schallhart, "A Brief Account of Runtime Verifica-
7. LLM:neg(exec_refund) U (mgr_approval) Human:[Approve/Reject/Edit] tion," Journal of Logic and Algebraic Programming, vol. 78, no. 5, pp. 293–303, 2009.
8. A. d'Avila Garcez and L. C. Lamb, "Neurosymbolic AI: The 3rd Wave,"
9. arXiv preprint arXiv:2012.05876, 2019.
10. S. Harnad, "The Symbol Grounding Problem," Physica D: Nonlinear Phenomena, vol. 42, no. 1-3, pp. 335–346, 1990.
11. N. F. Liu et al., "Lost in the Middle: How Language Models Use Long Contexts," Transactions of the Association for Computational Linguistics, vol. 12, pp. 157–173, 2024.
12. G. De Giacomo and M. Y. Vardi, "Linear Temporal Logic and Linear Dynamic Logic on Finite Traces," in Proc. of IJCAI, 2013, pp. 854–860.
13. G. De Giacomo et al., "ltlf2dfa: From LTLf to DFA," GitHub Repository, 2018. [Online]. Available: https://github.com/whitemech/ltlf2dfa
14. N. Shinn et al., "Reflexion: Language Agents with Verbal Reinforcement Learning," arXiv preprint arXiv:2303.11366, 2023.
15. OpenAI, "GPT-4 Technical Report," arXiv preprint arXiv:2303.08774, 2023.
16. J. Mao et al., "The Neuro-Symbolic Concept Learner: Interpreting Scenes, Words, and Sentences from Natural Supervision," in Proc. of ICLR, 2019.
17. A. Vaswani et al., "Attention is All You Need," in Adv. in Neural Information Processing Systems, vol. 30, 2017.
18. J. Ruan et al., "ToolEmu: A Framework for Automated Safety Test-ing of Large Language Models as Tool Agents," arXiv preprint arXiv:2309.15817, 2023.

ARTIFACTS

All source code, datasets, LTL specifications, and experimental scripts can be found at:

https://github.com/OMx0777/LogicGuard.

The repository includes :

- Complete implementation of LogicGuard (3,200 LoC)
- 2,400 test cases with ground truth annotations
- Red Team Adversarial Prompt Dataset.
- Jupyter notebooks reproducing all figures and tables
- Docker container for reproducible evaluation