NEXUS: Structured Runtime Safety for Tool-Using LLM Agents

Elias Hossain[1,†*], **Md. Mehedi Hasan Nipu**[2,†], **Tasfia Nuzhat Ornee**[1], **Rajib Rana**[3], **Niloofar Yousefi**[1]

[1]University of Central Florida [2]North South University
[3]University of Southern Queensland

Abstract

Tool-using LLM agents increasingly execute high-impact actions, making runtime safety monitoring essential. We present NEXUS (**Neural EXecution Utility and Safety**), a structured-plan safety monitor that applies a formal intervention policy Π to select among four actions: *allow*, *block*, *request confirmation*, or *request revision*. NEXUS combines deterministic safety rules, argument-level inspection, and a calibrated logistic-regression risk score for graded escalation. On a 128-instance synthetic benchmark, NEXUS attains $F_1 = 0.949$ and 4-class intervention accuracy of 0.6406, outperforming rule-only intervention selection by $+27.3$ percentage points. It also improves over rule-only on R-Judge ($F_1 = 0.861$ vs. 0.849), matches rule-only on AgentHarm due to threat-model limits, and achieves 0% ASR at 99% control allow on IPI. On the rule-blind NEXUS-Stress benchmark, NEXUS reaches $F_1 = 0.897$ with REVISE recall of 0.500, highlighting both progress and remaining difficulty in fine-grained intervention routing. With 0.38 ms median latency, NEXUS adds under 0.1% overhead to typical agent loops. Code, benchmarks, and the calibrated risk scorer are publicly released.[1]

1 Introduction

Large language models (LLMs) are increasingly deployed as tool-using agents that invoke APIs, access files, query databases, execute code, and act on behalf of users (OpenAI, 2024; Anthropic, 2024; He et al., 2025; Deng et al., 2025). This shift from passive text generation to action-oriented autonomy changes the safety problem: unsafe behavior can now cause destructive file operations, credential exposure, unauthorized network access, or resource abuse *before* a human notices (Yao et al., 2024; Das et al., 2025; Zhang et al., 2025).

The tool-use threat surface. Unsafe behavior arises from the agent's execution plan itself: the tools it selects, the inputs it provides, the permissions it requests, and the side effects it produces. An agent may read a sensitive file and then send its contents to an external location—each step reasonable alone, but together forming a data-leakage plan. Prompt injection, malicious retrieved content, corrupted tool outputs, or ambiguous requests can also redirect agents toward unsafe behavior (Perez and Ribeiro, 2022; Greshake et al., 2023; Zhan et al., 2024). Agent safety thus requires monitoring *planned tool use*, not only generated text (Bodea et al., 2025; Maloyan and Namiot, 2026).

Limits of existing safeguards. Existing safeguards address parts of this problem. Declarative rule systems (Wang et al., 2025; Kamath et al., 2025) enforce predefined constraints, symbolic verifiers (Miculicich et al., 2025; Zhao et al., 2026) inspect individual tool calls, and judge-style filters (Yuan et al., 2024) evaluate trajectories. Yet these approaches operate at a single abstraction level, lack plan-level escalation, or reduce safety to a binary decision. In practice, some plans should be blocked, some should require confirmation, and others revised rather than fully rejected.

We present NEXUS, a structured runtime safety monitor for tool-using language agents. It operates over a plan representation containing ordered tool invocations, arguments, permissions, side-effect categories, sensitivity and irreversibility flags, and resource estimates. Given a proposed plan, NEXUS selects one of four interventions: *allow*, *block*, *request confirmation*, or *request revision*. Its hybrid backbone combines deterministic rules, argument-level inspection, and calibrated risk scoring; the learned component does not replace

*Corresponding author: mdelias.hossain@ucf.edu.
†Equal contribution.
[1]https://github.com/eliashossain001/nexus

Findings of the Association for Computational Linguistics: IJCNLP-AACL 2026, pages 2203–2232
November 6-10, 2026 ©2026 Association for Computational Linguistics

the rule layer but supplies a calibrated escalation signal for fine-grained routing while preserving auditability.

We evaluate NEXUS across held-out synthetic tests, adversarial indirect-prompt-injection settings, multi-turn escalation scenarios, R-Judge, AgentHarm, and the rule-blind NEXUS-Stress benchmark. NEXUS achieves strong structured-plan detection and improves fine-grained intervention routing over rule-only monitoring. Plan-level monitoring is effective when unsafe behavior is visible in tool actions, arguments, permissions, or side effects, but reaches a boundary when harmful and benign behaviors share the same structural plan and differ mainly in latent intent, where prompt- or intent-aware safeguards remain necessary (Zhong et al., 2025). Our main contributions are:

- **Structured runtime safety.** We introduce **NEXUS**, a plan-level safety monitor that inspects tool-using LLM agents *before* tool execution.
- **Graded intervention policy.** We formulate a four-action policy that routes plans to *allow*, *block*, *request confirmation*, or *request revision*, going beyond binary safety decisions.
- **Hybrid and interpretable detection.** We develop an interpretable framework that combines deterministic rules, argument-level inspection, and calibrated risk scoring under a unified intervention policy.
- **Comprehensive evaluation and release.** We evaluate NEXUS across synthetic, adversarial, multi-turn, external, and rule-blind benchmarks, and release the code, benchmarks, trained scorer, calibration metadata, and reproducibility scripts.

2 Related Work

Runtime safety for tool-using agents. Prior work approaches agent safety along three axes: declarative rule systems (Wang et al., 2025; Kamath et al., 2025) encode invariants over agent behaviour; symbolic verifiers (Miculicich et al., 2025; Zhao et al., 2026) check individual tool calls; and judge-style monitors (Yuan et al., 2024) score whole trajectories with another LLM. Orthogonal benchmarks (ToolSafe (Mou et al., 2026), AgentHarm (Andriushchenko et al., 2025), Wu et al. (Wu et al., 2026)) target tool-use harms, harmful behaviours, and trace-perturbation robustness; the Verifier-Tax study (Sah et al., 2026) analyses the cost of always-on verification.

Prompt-injection defences. A complementary line targets prompt-injection at the input/model layer. StruQ (Chen et al., 2025) fine-tunes a structured-query interface separating trusted instructions from untrusted data; Spotlighting (Hines et al., 2024) marks untrusted content via delimiting, datamarking, or encoding. Both operate before any tool call is committed and are orthogonal to NEXUS, which monitors the resulting plan regardless of the upstream defence: once an injection steers the agent, the plan is caught by any structured-plan monitor (Appendix H).

Positioning. We position NEXUS (Table 1) at the intersection of three design choices: (i) monitoring at the granularity of a structured plan IR with argument-level inspection; (ii) a hybrid of deterministic rules, calibrated learned risk, and argument inspection unified by a formal four-class intervention policy Π; and (iii) an explicit safety–utility objective (Eq. 2) that makes the threshold trade-off tunable.

3 Threat Model

We consider an adversary who can (i) issue a directly malicious request, (ii) inject instructions into retrieved documents or prior tool outputs (*IPI*), (iii) return adversarial tool outputs, (iv) exploit permission mismatches (*privilege misuse*), (v) chain individually safe steps into multi-turn exfiltration (*long-horizon escalation*), or (vi) cause the agent to exceed its resource budget. The adversary cannot modify the LLM weights, safety rules, or tool implementations. For each plan P, the defender selects $\Pi(P) \in \{\text{ALLOW}, \text{BLOCK}, \text{CONFIRM}, \text{REVISE}\}$ to minimize the safety–utility loss in Eq. 2.

4 The NEXUS Framework

Three signals, one policy

Rules, argument inspection, and a calibrated scorer are independent and testable in isolation; every Π decision is traceable to a named rule, an argument finding, or a threshold crossing.

Figure 1 summarizes the NEXUS pipeline. The agent produces a structured plan—selected tools, arguments, permissions, side-effect categories, execution context—rather than executing tool calls

System	Plan IR	Rules	Learned	Arg.	Session	Adv.	Multi-turn	Compare-to-NEXUS
AgentSpec (Wang et al., 2025)	declarative	✓	✗	*part.*	✗	*part.*	✗	baseline
VeriGuard (Miculicich et al., 2025)	symbolic	✓	✗	✓	✗	✗	*part.*	conceptual
ToolSafe (Mou et al., 2026)	trajectory	*part.*	✓	*part.*	✓	*part.*	✓	benchmark
CLAWGUARD (Zhao et al., 2026)	per-call	✓	✗	✓	✗	✗	✗	conceptual
Agent-C (Kamath et al., 2025)	constitution	✓	✗	✗	✗	✗	*part.*	conceptual
Verifier-Tax (Sah et al., 2026)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	motivation
AgentHarm (Andriushchenko et al., 2025)	N/A	N/A	N/A	N/A	N/A	✓	*part.*	benchmark
R-Judge (Yuan et al., 2024)	judge model	✗	✓	*part.*	*part.*	*part.*	✓	benchmark+baseline
Wu et al. (Wu et al., 2026)	trace perturb.	✗	✗	✗	✗	✓	✓	motivation
NEXUS	**structured plan IR**	✓	**calib.**	✓	✓	✓	*part.*	–

Table 1: NEXUS vs. prior runtime-safety systems and benchmarks. ✓ = full support; *part.* = partial. *Compare-to*-NEXUS: **baseline** (empirical baseline), **conceptual** (design-level only), **benchmark+baseline** (both), **motivation** (cited for framing only). *Multi-turn* is *part.* because the single-turn policy reaches $F_1 = 0.500$ on *multi_step_escalation*; the session-aware extension (§7.2) mitigates this on the authored 120-session benchmark but is not validated on organic traces.

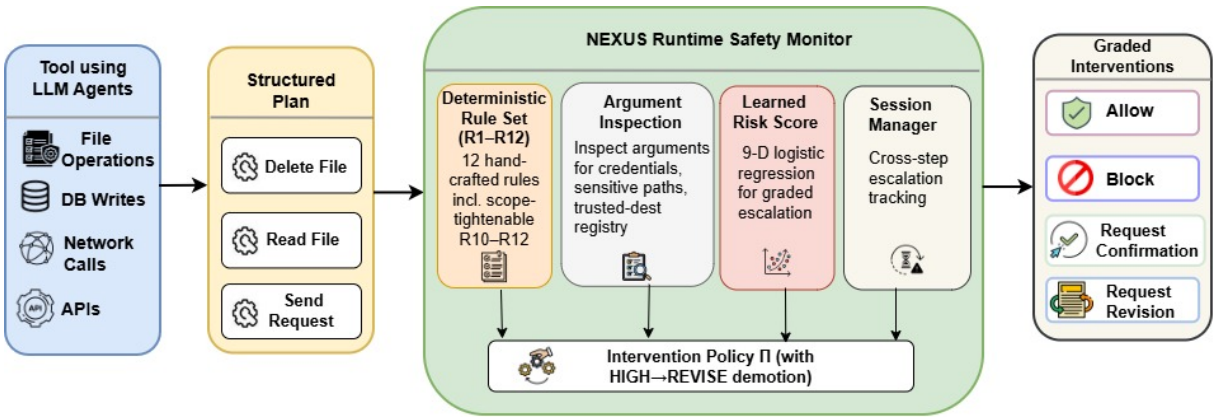


Figure 1: Overview of NEXUS. **(a)** A tool-using LLM agent emits a **(b)** *structured plan* which **(c)** NEXUS evaluates through four components (deterministic rules, argument inspection, calibrated risk ρ, session manager). **(d)** A unified policy Π routes each plan to one of four *graded interventions*: ALLOW, BLOCK, CONFIRM, or REVISE.

directly. NEXUS intercepts the plan and evaluates it through four complementary components: a deterministic rule set for explicit violations, an argument inspector for sensitive values, a calibrated learned risk scorer for graded escalation, and a session manager for cross-turn state. The resulting signals feed an intervention policy that decides whether to ALLOW, BLOCK, request user CONFIRMATION, or request plan REVISION. The remainder of this section defines each component and the policy (§4.3–§4.4).

4.1 Structured Plan IR

A plan is $P = (s_1, \ldots, s_n, c)$, where each step s_i specifies a tool name, arguments, side-effect category (file/database/network/compute), irreversibility flag, sensitivity flag, required permissions, and estimated cost. The context c records the agent's available permissions, remaining budget, prior tool-output history, and per-session policy flags.

4.2 Rules and Argument Inspection

NEXUS applies twelve deterministic rules. The first six cover permission checks, irreversible actions, sensitive access, network access, budget limits, and suspicious patterns. Rules R7–R9, added during development, address scope-ambiguous actions (security-posture changes; severity HIGH/CRIT), broad-scope operations (wildcards/globs; severity MED), and external-source-driven privileged actions (severity HIGH). Three further rules (R10–R12) target medium-severity REVISE-worthy patterns identified through error analysis on NEXUS-Stress: R10, *bulk audience operation*, flags plans sending to unbounded audiences; R11, *high-cost batch*, flags expensive batch processing over large datasets; and R12, *global config rollout*, flags system-wide configuration changes without staged deployment.

Each rule emits violations with severity $\mathrm{sev}(v) \in \{\text{LOW}, \text{MED}, \text{HIGH}, \text{CRIT}\}$. In parallel, the argument inspector scans step arguments for

Threat	Channel	NEXUS handling
Direct malicious request	User prompt	Rules + arg. + ρ on plan
Indirect prompt injection	RAG/tool output	Structured-plan monitoring
Corrupted tool output	API/DB response	Schema + suspicious-pattern rules
Privilege misuse	Plan execution	Permission rule
Sensitive exfiltration	Network step	Argument inspection
Resource abuse	Budget overrun	Budget rule
Long-horizon escalation	Multi-turn session	*Single-turn: partial;* session-aware Π mitigates on authored benchmark (§7.2)
Jailbreaking agent LLM	Model weights	*Out of scope*
Tool-side compromise	Tool impl.	*Out of scope*
Covert side channels	Timing/cache	*Out of scope*

Table 2: Threats addressed by NEXUS at runtime. The single-turn policy detects $F_1 = 0.500$ on *multi_step_escalation*; the session-aware extension of §7.2 mitigates this on the 120-session authored benchmark but is not yet validated on organic multi-turn traces (§9).

credential patterns, sensitive paths, sensitive tables or fields, and suspicious URLs. These argument-level findings are treated as violations on equal footing with rule-based violations V_R. This component improves the *sensitive data access* category from $F_1 = 0.273$ to 1.000 (Appendix L).

4.3 Calibrated Learned Risk

Feature vector and calibration. We train a logistic-regression risk scorer on a 9-D plan-level feature vector $\phi(P)$ (plan length; irreversible/sensitive/delete-step indicators; per-category step counts; total cost; tool-category diversity; Appendix B). A stratified $70/15/15$ split (seeds $42/7$) of the 428-instance synthetic benchmark yields train, validation, and test sets; thresholds (τ_b, τ_c) are selected on validation and all reported metrics come from the held-out test split exclusively. Validation-best thresholds are $\tau_b = 0.30$, $\tau_c = 0.10$; the paper defaults $(0.75, 0.70)$ yield identical F_1 with near-identical test intervention accuracy (0.641 vs. 0.651), confirming no threshold leakage. Splitting the training set $80/20$ (seed 7) into base-fit and calibration sets, Platt scaling reduces ECE $\approx 6.5\times$ over the raw scorer (Table 9, Appendix C); we use Platt-scaled ρ throughout.

Role in Π. We use ρ as a graded escalation signal rather than a primary detection engine. Its specific role is the scorer-gated demotion step in Π (§4.4): when exactly one critical rule fires, ρ determines whether to issue a hard block or route the case to user confirmation, improving 4-class intervention accuracy by $+27.3$ pp (permutation $p < 0.001$). The calibration analysis ensures that (τ_b, τ_c) operate on a probability scale reflecting empirical unsafe rates.

4.4 Intervention Policy

Let $V(P) = V_R(P) \cup V_A(P)$ be the combined violation set and let $n_{\text{crit}}(P) = |\{v \in V(P) : \text{sev}(v) = \text{CRIT}\}|$. With thresholds $(\tau_b, \tau_c) = (0.75, 0.70)$, the scorer-gated demotion policy is

$$\Pi(P) = \begin{cases} \text{BLOCK} & n_{\text{crit}} \geq 2 \\ \text{BLOCK} & n_{\text{crit}} = 1 \wedge \rho(P) \geq \tau_b \\ \text{CONFIRM} & n_{\text{crit}} = 1 \wedge \rho(P) < \tau_b \\ \text{REVISE} & \exists\, v : \text{sev}(v) = \text{HIGH} \wedge \textit{scope-tight.} \wedge \rho < \tau_s \\ \text{CONFIRM} & \exists\, v : \text{sev}(v) = \text{HIGH} \\ \text{REVISE} & \exists\, v : \text{sev}(v) = \text{MED} \\ \text{CONFIRM} & \rho(P) \geq \tau_c \\ \text{ALLOW} & \text{otherwise} \end{cases} \tag{1}$$

Cases are evaluated top-to-bottom, returning the first match. Multiple critical violations trigger an unconditional block. When exactly one critical violation is present, the risk score gates block ($\rho \geq \tau_b$) versus confirmation ($\rho < \tau_b$)—the scorer-gated demotion step. A HIGH→REVISE demotion fires when all high-severity violations come from *scope-tightenable* rules (e.g., `network_access_control`, `permission_check`) and $\rho < \tau_s = 0.50$, routing the plan to revision rather than confirmation. Otherwise, high-severity violations route to confirmation, medium-severity to revision, and the scorer provides a residual confirmation signal when no rule violation is present.

Safety–utility objective. The thresholds (τ_b, τ_c) are tunable parameters of Π. For a workload $\mathcal{W}$ of (plan, gold-label) pairs, we define

$$\mathcal{L}(\Pi) = \mathbb{E}_{(P,y)\sim\mathcal{W}}[\lambda_s\, u_s + \lambda_o\, u_o + \lambda_c\, c(\Pi(P))] \tag{2}$$

where $u_s = \mathbf{1}[y = \text{UNSAFE} \wedge \Pi(P) = \text{ALLOW}]$ is the unsafe-allow indicator, $u_o = \mathbf{1}[y = \text{SAFE} \wedge \Pi(P) \neq \text{ALLOW}]$ is the overblocking indicator, and $c(\cdot)$ is the per-intervention cost: $c(\text{ALLOW}) = 0$, $c(\text{REVISE}) = 0.1$, $c(\text{CONFIRM}) = 0.3$, and $c(\text{BLOCK}) = 1$. Section E shows that binary F_1 and the unsafe-caught rate are invariant to (τ_b, τ_c) on this distribution; only the 4-class intervention accuracy is sensitive to the operating point.

Algorithm. The full procedure (Appendix A) computes V_R, V_A, and ρ as logically independent signals, then dispatches via Eq. 1. Median CPU latency is 0.38 ms per plan on the IPI set (∼2,600 decisions/sec); the dominant cost is `predict_proba`.

5 NEXUS-Bench

We evaluate NEXUS on two distributions.

Synthetic split. NEXUS-Bench v1 contains 428 instances from 9 deterministic templates over 7 tool categories. A stratified 70/15/15 split (seed 42) yields 300/63/128 train/validation/test sets covering nine risk categories: *safe*, *destructive action*, *sensitive data access*, *external communication*, *privilege misuse*, *budget abuse*, *ambiguous confirm*, *ambiguous revise*, and *multi-step escalation*. Each instance carries a natural-language request, an ordered structured plan, a binary safety label, and an intervention label in {ALLOW, BLOCK, CONFIRM, REVISE}.

IPI adversarial split. We construct 200 paired indirect-prompt-injection cases across three templates (Section 3): (A) injection in a retrieved RAG chunk (40 pairs), (B) injection in an earlier tool output (40 pairs), and (C) a plan-level evasion annotation claiming pre-approval (20 pairs). Each adversarial case has a matched benign control (same topic, user, tool category) without the unsafe step, testing whether NEXUS correctly classifies the agent's plan when the surrounding *context* contains adversarial text.

The two synthetic splits evaluate structural detection and injection-context robustness. External generalization is evaluated on R-Judge (Yuan et al., 2024) (§7.3) and AgentHarm (Andriushchenko et al., 2025) (§7.4); organic-trace evaluation is left for future work (§9).

6 Main Results

Table 3 summarizes results across the held-out synthetic split, R-Judge, and the indirect-prompt-injection (IPI) split. Synthetic measures in-distribution structured-plan detection; R-Judge is the main external validation. IPI is a robustness check: both NEXUS and the rule-only monitor block all adversarial plans, showing that plan-level monitoring catches injected behavior once it appears in the structured plan. AgentHarm is reported separately in §7.4 because harmful and benign examples may share the same target tools.

	Synthetic (n=128)		R-Judge (n=564)		IPI (n=200)	
Monitor	F_1	**Int.-Acc**	F_1	**Prec.**	F_1	**Int.-Acc**
Rule-only	0.949	0.367	0.849	0.858	1.000	1.000
Learned-only	0.549	0.195	–	–	1.000	1.000
NEXUS	**0.949** [.92,.98]	**0.641** [.56,.72]	**0.861** [.83,.89]	**0.858** [.82,.90]	0.995 [.98,1.00]	**0.995**

Table 3: Headline results across synthetic, R-Judge, and IPI settings. Bracketed values are 95% bootstrap CIs for NEXUS. Synthetic and R-Judge measure in-distribution and external performance, respectively; IPI is a robustness check for plan-level monitoring under injected context.

On the synthetic split, NEXUS achieves $F_1 = 0.949$ and 4-class intervention accuracy 0.641: identical binary F_1 as rule-only, with routing improved from 0.367 to 0.641. On R-Judge, NEXUS reaches $F_1 = 0.861$ $[0.83, 0.89]$ vs. 0.849 rule-only, driven by additional unsafe detections, especially IoT after enabling the permanence-flag rule (§7.3). Per-category R-Judge is strongest for Finance, Program, and Web ($F_1 \geq 0.89$), followed by Application (0.852); IoT remains hardest (0.518).

On AgentHarm (n=352), both NEXUS and rule-only obtain $F_1 = 0.591$ $[0.53, 0.65]$: paired harmful and benign examples share target tools, so the structural plan lacks the intent signal. We treat AgentHarm as a threat-model boundary rather than a regression.

Statistical significance. Binary McNemar gives $p = 1.0$ on the synthetic split (identical binary decisions), so the scorer's effect concentrates in 4-class intervention accuracy. A paired permutation test (5,000 permutations) gives $\Delta = +0.273$, $p < 0.001$. On R-Judge, McNemar yields $\chi^2 = 3.125$, $p = 0.077$ (+7 unsafe detections).

Scorer-gated demotion adds +27.3 pp

4-class intervention accuracy rises 0.367 → 0.641 ($p < 0.001$, permutation) at identical binary $F_1 = 0.949$ and unchanged unsafe-caught rate.

7 Extended Empirical Analysis

7.1 Category-Level Behavior

Table 16 reports binary F_1 by risk category on the synthetic test split. NEXUS gains most when unsafe behavior appears inside tool arguments or plan structure. Versus rule-only, NEXUS improves *sensitive data access* from $F_1 = 0.273$ to 1.000: argument inspection catches credentials and sensitive

Metric	No session memory	With session memory
TP / FN (critical turn)	0 / 95	**95 / 0**
FP / TN (legitimate ctrl)	0 / 25	**0 / 25**
Precision / Recall / F_1	0.00 / 0.00 / 0.00	**1.00 / 1.00 / 1.00**†
Control allow rate	1.000	1.000‡

Table 4: Session-aware multi-turn evaluation on 120 authored sessions. Session memory catches all 95 unsafe critical turns with zero false positives on 25 legitimate controls. †Wilson 95% CI for $95/95$: $[0.961, 1.000]$. ‡Wilson 95% CI for $25/25$: $[0.864, 1.000]$.

paths that category-level rules miss. Versus learned-only, NEXUS improves *budget abuse*, *ambiguous confirm*, and *ambiguous revise* by $+1.000$ each, showing that a scalar risk score alone misses structural signals such as exceeded budgets or underspecified intent. The main remaining weakness is *multi-step escalation*, where single-turn evaluation reaches only $F_1 = 0.500$.

7.2 Session-Aware Multi-Turn Monitoring

The low score on *multi-step escalation* comes from evaluating each turn independently. We test whether session context resolves this on a 120-session benchmark: cross-step exfiltration (35), escalating privilege (20), incremental sensitive access (20), repeated unsafe intent (20), and legitimate multi-turn controls (25).

We extend the CONFIRM branch of Π with four session-state features (exposed entities, cumulative permissions, repeated-unsafe-attempt counter, cumulative cost); no additional learned component is added. Session memory raises critical-turn detection from $0/95$ to $95/95$ with zero false positives on the 25 legitimate controls (Table 4), showing that the failure was missing session state rather than rule vocabulary.

7.3 External Generalization on R-Judge

Benchmark setup. We evaluate external generalization on R-Judge (Yuan et al., 2024), a public multi-turn agent-safety benchmark with 564 usable records across Application, Finance, IoT, Program, and Web, after excluding seven leaked records. Each record provides an agent–environment trajectory, binary safety label, and attack type. We parse

Subset	Monitor	F_1 [95% CI]	Prec.	Rec.
Full (n=564)	Rule-only	0.849	0.858	0.841
Full (n=564)	NEXUS	**0.861**		
[0.831, 0.888]	0.858			
[0.821, 0.895]	0.864			
[0.823, 0.901]				
Real-plan (n=491)	NEXUS	0.832	0.831	0.834

Table 5: External generalization on R-Judge. NEXUS improves over the rule-only monitor on the full benchmark, increasing recall while preserving precision. The real-plan subset includes only records that could be parsed into structured tool plans, excluding dialogue-only placeholder cases.

structured `agent.action` strings into NEXUS `PlanStep` objects; $491/571$ records yield real structured plans, while 80 dialogue-only Application records map to placeholder steps. Tool metadata is inferred via keyword tables.

Results. Table 5 shows NEXUS reaches $F_1 = 0.861$, $+0.012$ over rule-only, while below the synthetic result (0.949) as expected under distribution shift. The gain comes from seven additional unsafe detections (McNemar $\chi^2 = 3.125$, $p = 0.077$). Retraining the scorer on R-Judge with 5-fold CV raises F_1 to 0.892 vs. 0.848 synthetic-trained, indicating transferable safety signal in the feature representation.

Category analysis. NEXUS performs best on Finance, Program, and Web, where risks appear as parseable transfer, delete, transmit, or execution actions. IoT remains hardest: the permanence-flag rule lifts IoT F_1 from 0.462 to 0.518 and overall R-Judge F_1 from 0.848 to 0.861, but remaining errors involve cross-step identity matching and incomplete smart-home semantics. R-Judge thus supports external validity for structured tool-call monitoring while highlighting the need for richer domain semantics.

7.4 Threat-Model Boundary on AgentHarm

AgentHarm (Andriushchenko et al., 2025) contains 176 harmful and 176 benign tool-use behaviors across eight categories, each with a free-text prompt and target functions. We map each target function to a NEXUS `PlanStep` using the R-Judge keyword classifier, without including the prompt text in the plan.

On this 352-instance set, NEXUS obtains $F_1 = 0.591$ $[0.53, 0.65]$ (precision 0.508, recall 0.705); rule-only scores identically. AgentHarm pairs

Metric	Value
Median latency	0.205 **ms**
P90 latency	0.324 ms
P95 latency	0.362 ms
P99 latency	0.456 ms
Throughput	∼4,349 **decisions/sec**

Table 6: Runtime latency of NEXUS on the synthetic test split. Results are computed over 384 evaluations from three repeated runs on a single CPU core without batching.

harmful and benign examples that share target tools, so the unsafe signal lies in prompt intent rather than the structural tool sequence.

Scope analysis. AgentHarm thus tests a boundary of plan-level monitoring. NEXUS detects unsafe *structure* in plans, arguments, permissions, and side effects, not latent harmful intent in prompts. When harmfulness depends on prompt-level intent while the tool plan is structurally similar, prompt- or argument-aware methods such as ToolSafe (Mou et al., 2026) are the more appropriate comparison class. NEXUS matching rule-only here reinforces the design claim: the learned risk score routes borderline interventions rather than detecting latent intent.

7.5 Runtime Efficiency

Table 6 reports runtime cost on the 128-instance synthetic test split, aggregated over three runs (384 decisions) on a single CPU core without batching. NEXUS achieves 0.205 ms median and 0.456 ms P99 latency (≈ 4,349 decisions/sec)—negligible compared to a typical LLM agent turn of hundreds of milliseconds. It can thus be inserted before tool execution without a GPU or noticeable slowdown.

7.6 Comparison with LLM Judges

We compare NEXUS with GPT-4o as a strong LLM-judge baseline on the 200-instance NEXUS-Stress benchmark. GPT-4o obtains higher binary and intervention accuracy but at substantially higher latency (Table 7): NEXUS reaches 94% of its binary F_1 and 86% of its intervention accuracy while running ≈ 3,200× faster. GPT-4o also struggles with fine-grained routing (per-class F_1: 0.455 CONFIRM, 0.405 REVISE), suggesting the hardest part of NEXUS-Stress is deciding the correct intervention level, not only detecting unsafe plans.

Monitor	Binary F_1	Int.-Acc	Latency
GPT-4o	0.956	0.555	∼1,200 ms
NEXUS	**0.897**	**0.480**	**0.38 ms**
NEXUS / GPT-4o	94%	86%	∼3,200× **faster**†

Table 7: Comparison with GPT-4o on the 200-instance NEXUS-Stress benchmark. NEXUS is less accurate than GPT-4o but is several orders of magnitude faster, making it more suitable for runtime monitoring before tool execution. †Speed ratio uses a conservative 0.38 ms NEXUS latency.

gold ↓ / pred →	ALLOW	BLOCK	CONFIRM	REVISE
ALLOW ($n = 35$)	**8**	2	9	16
BLOCK ($n = 71$)	1	**38**	19	13
CONFIRM ($n = 54$)	5	5	**30**	14
REVISE ($n = 40$)	3	9	8	**20**

Table 8: NEXUS-Stress 4-class confusion matrix (after R10–R12 and HIGH→REVISE demotion). REVISE recall improves from 0/40 to 20/40 (0.500). The remaining errors are dominated by over-blocking of safe (ALLOW) plans, which are escalated to CONFIRM or REVISE, and by BLOCK/CONFIRM/REVISE confusion among unsafe plans.

7.7 Adversarial Stress Testing

We evaluate NEXUS on **NEXUS-Stress**, a 200-example adversarial benchmark authored without matching the rule-set vocabulary, covering 20 scenario categories across 11 domains (Appendix P).

NEXUS achieves binary $F_1 = 0.897$ (precision 0.852, recall 0.945). Three targeted medium-severity rules (R10–R12) and the scorer-gated HIGH→REVISE demotion for scope-tightenable violations improve REVISE recall from 0/40 to 20/40 (0.500), raising 4-class intervention accuracy from 0.435 to 0.480 (confusion matrix: Table 8). The demotion fires when all high-severity violations come from scope-tightenable rules (e.g., `network_access_control`, `permission_check`) and $\rho < 0.50$, marking plans fixable by narrowing scope rather than requiring full human authorization.

REVISE recall remains at 0.500: cases where the fixable pattern is expressed through critical-severity rules (e.g., exfiltration to a trusted auditor) still route to BLOCK, CONFIRM, or ALLOW (§9).

7.8 Additional Results in the Appendix

The appendix reports supporting analyses: confusion matrices (Appendix D), threshold and safety–

utility sweeps (Appendix E), strengthened learned-only baselines (Appendix F), policy ablations (Appendix G), IPI v2 across five injection styles (Appendix H), scorer-based reclassification (Appendix J), error analysis (Appendix K), and the full NEXUS-Stress breakdown (Appendix P). Key findings are summarized in Tables 3, 5, and 8.

8 Discussion

Runtime safety for tool-using agents is best viewed as a plan-level control problem, where risks emerge through tool choices, arguments, permissions, side effects, and execution order. NEXUS inspects this structured plan before tool calls are executed and grounds interventions in rules, argument findings, or calibrated risk scores. The results show that the hybrid design is effective: rules enforce hard constraints, argument inspection detects sensitive inputs, and the calibrated scorer improves BLOCK versus CONFIRM routing. The main challenge is fine-grained intervention selection. Deciding whether to BLOCK, CONFIRM, or REVISE remains difficult, especially for medium-severity and rule-blind cases. R-Judge and AgentHarm results show that NEXUS works best when risk is visible in executable structure, such as deletion, transmission, sensitive access, or permission misuse. It is weaker when harmfulness depends on latent prompt intent, so it should complement prompt-aware safeguards. Finally, sub-millisecond CPU latency makes NEXUS practical for runtime governance before tool execution, while its modular design supports easy updates to rules, checkers, and thresholds.

9 Conclusion

We presented NEXUS, a runtime safety monitor for tool-using LLM agents that combines twelve deterministic rules, argument-level inspection, and a calibrated risk scorer under a four-class scorer-gated demotion policy Π. The scorer is used for intervention selection rather than binary detection, and a new HIGH$\rightarrow$REVISE demotion path routes scope-tightenable violations to revision when the scorer indicates low risk. Across benchmarks, NEXUS achieves strong safety performance with practical runtime overhead. On the 128-instance synthetic split, it obtains $F_1 = 0.949$ and 4-class intervention accuracy of 0.641. On R-Judge (n=564), it reaches $F_1 = 0.861$ $[0.83, 0.89]$, improving over rule-only monitoring. On AgentHarm, NEXUS matches rule-only performance ($F_1 = 0.591$), highlighting a deliberate threat-model boundary. On NEXUS-Stress, NEXUS reaches $F_1 = 0.897$ with REVISE recall of 0.500 (improved from $0/40$ to $20/40$ via three new medium-severity rules and scorer-gated demotion), while requiring only 0.38 ms median latency. An online agent integration evaluation (n=30) achieves perfect binary detection ($F_1 = 1.0$) at 90% 4-class accuracy, and inter-annotator agreement on NEXUS-Stress labels reaches Cohen's $\kappa = 0.688$ (substantial). Future work should further expand medium-severity coverage and validate on organic agent traces. We release the code, benchmarks, trained scorer, and CI pipelines.

Limitations and Ethical Considerations

Scope. Several evaluations use author-generated templates, so in-distribution results should be viewed as upper bounds rather than deployment estimates. R-Judge ($F_1 = 0.861$) and NEXUS-Stress ($F_1 = 0.897$) provide stronger out-of-distribution evidence, but live-trace evaluation is still needed. An online agent integration evaluation (n=30, Appendix W) achieves perfect binary detection, providing initial end-to-end evidence. NEXUS currently supports 30+ tools via its tool permission registry; broader registries require new side-effect and permission encodings. Human evaluation of explanations/revisions is also left for future work.

Coverage gap. NEXUS-Stress shows that rule-blind medium-severity cases remain challenging. After adding three targeted rules (R10–R12) and scorer-gated HIGH$\rightarrow$REVISE demotion, REVISE recall improves from $0/40$ to $20/40$ (0.500). The remaining 20 missed cases involve scope-tightening patterns expressed through critical-severity rules that currently route to BLOCK.

Intervention difficulty. CONFIRM/REVISE routing remains challenging, with even GPT-4o reaching only 0.555 intervention accuracy on adversarial data. Hybrid fast-filter and LLM-judge designs may help.

Ethics. The benchmarks contain no real PII and use fabricated credentials/URLs. We release templates following prior benchmark norms, as exposing failure modes primarily supports defensive research.

References

Maksym Andriushchenko, Alexandra Souly, Mateusz Dziemian, Derek Duenas, Maxwell Lin, Justin Wang, Dan Hendrycks, Andy Zou, Zico Kolter, Matt Fredrikson, and 1 others. 2025. Agentharm: A benchmark for measuring harmfulness of llm agents. In *International Conference on Learning Representations*, volume 2025, pages 79185–79220.

Anthropic. 2024. Model Context Protocol. `https://modelcontextprotocol.io/`. Specification and documentation.

Teofil Bodea, Masanori Misono, Julian Pritzi, Patrick Sabanic, Thore Sommer, Harshavardhan Unnibhavi, David Schall, Nuno Santos, Dimitrios Stavrakakis, and Pramod Bhatotia. 2025. Trusted ai agents in the cloud. *arXiv preprint arXiv:2512.05951*.

Sizhe Chen, Julien Piet, Chawin Sitawarin, and David Wagner. 2025. Struq: Defending against prompt injection with structured queries. In *USENIX Security Symposium*.

Badhan Chandra Das, M Hadi Amini, and Yanzhao Wu. 2025. Security and privacy challenges of large language models: A survey. *ACM Computing Surveys*, 57(6):1–39.

Zehang Deng, Yongjian Guo, Changzhou Han, Wanlun Ma, Junwu Xiong, Sheng Wen, and Yang Xiang. 2025. Ai agents under threat: A survey of key security challenges and future pathways. *ACM Computing Surveys*, 57(7):1–36.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 2023. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM workshop on artificial intelligence and security*, pages 79–90.

Feng He, Tianqing Zhu, Dayong Ye, Bo Liu, Wanlei Zhou, and Philip S Yu. 2025. The emerged security and privacy of llm agent: A survey with case studies. *ACM Computing Surveys*, 58(6):1–36.

Keegan Hines, Gary Lopez, Matthew Hall, Federico Zarfati, Yonatan Zunger, and Emre Kiciman. 2024. Defending against indirect prompt injection attacks with spotlighting. *arXiv preprint arXiv:2403.14720*.

Adharsh Kamath, Sishen Zhang, Calvin Xu, Shubham Ugare, Gagandeep Singh, and Sasa Misailovic. 2025. Enforcing temporal constraints for llm agents. *arXiv preprint arXiv:2512.23738*.

Narek Maloyan and Dmitry Namiot. 2026. Breaking the protocol: Security analysis of the model context protocol specification and prompt injection vulnerabilities in tool-integrated llm agents. *arXiv preprint arXiv:2601.17549*.

Lesly Miculicich, Mihir Parmar, Hamid Palangi, Krishnamurthy Dj Dvijotham, Mirko Montanari, Tomas Pfister, and Long T Le. 2025. Veriguard: Enhancing llm agent safety via verified code generation. *arXiv preprint arXiv:2510.05156*.

Yutao Mou, Zhangchi Xue, Lijun Li, Peiyang Liu, Shikun Zhang, Wei Ye, and Jing Shao. 2026. Toolsafe: Enhancing tool invocation safety of llm-based agents via proactive step-level guardrail and feedback. *arXiv preprint arXiv:2601.10156*.

OpenAI. 2024. Swarm. `https://github.com/openai/swarm`. GitHub repository.

Fábio Perez and Ian Ribeiro. 2022. Ignore previous prompt: Attack techniques for language models. *arXiv preprint arXiv:2211.09527*.

Tanmay Sah, Vishal Srivastava, Dolly Sah, and Kayden Jordan. 2026. The verifier tax: Horizon dependent safety success tradeoffs in tool using llm agents. *arXiv preprint arXiv:2603.19328*.

Haoyu Wang, Christopher M Poskitt, and Jun Sun. 2025. Agentspec: Customizable runtime enforcement for safe and reliable llm agents. *arXiv preprint arXiv:2503.18666*.

Zekun Wu, Adriano Koshiyama, Sahan Bulathwela, and Maria Perez-Ortiz. 2026. Sell me this stock: Unsafe recommendation drift in llm agents. *arXiv preprint arXiv:2603.12564*.

Yifan Yao, Jinhao Duan, Kaidi Xu, Yuanfang Cai, Zhibo Sun, and Yue Zhang. 2024. A survey on large language model (llm) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing*, 4(2):100211.

Tongxin Yuan, Zhiwei He, Lingzhong Dong, Yiming Wang, Ruijie Zhao, Tian Xia, Lizhen Xu, Binglin Zhou, Fangqi Li, Zhuosheng Zhang, and 1 others. 2024. R-judge: Benchmarking safety risk awareness for llm agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 1467–1490.

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. 2024. Injecagent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 10471–10506.

Kaiyuan Zhang, Zian Su, Pin-Yu Chen, Elisa Bertino, Xiangyu Zhang, and Ninghui Li. 2025. Llm agents should employ security principles. *arXiv preprint arXiv:2505.24019*.

Wei Zhao, Zhe Li, Peixin Zhang, and Jun Sun. 2026. Clawguard: A runtime security framework for tool-augmented llm agents against indirect prompt injection. *arXiv preprint arXiv:2604.11790*.

Peter Yong Zhong, Siyuan Chen, Ruiqi Wang, McKenna McCall, Ben L Titzer, Heather Miller, and Phillip B Gibbons. 2025. Rtbas: Defending llm agents against prompt injection and privacy leakage. *arXiv preprint arXiv:2502.08966*.

Appendix

This appendix provides supplementary experimental results, per-category analyses, method details, deployment notes, and reproducibility, ethics, and AI-usage statements. The material is organized thematically to help readers locate supporting evidence for the main paper.

Algorithm and method details.

- **Appendix A** presents the complete runtime monitoring algorithm with scorer-gated demotion.
- **Appendix B** defines the 9-dimensional plan-level feature vector $\phi(P)$.
- **Appendix C** reports calibration metrics and the reliability diagram.

Extended experimental results.

- **Appendix D** reports binary and 4-class confusion matrices for NEXUS on the NEXUS-Stress benchmark.
- **Appendix E** reports the (τ_b, τ_c) threshold sweep and the 5×5 safety–utility sensitivity analysis.
- **Appendix F** reports strengthened learned-only baselines.
- **Appendix G** reports policy ablations with bootstrap confidence intervals.
- **Appendix H** reports IPI robustness across five injection styles.
- **Appendix I** provides qualitative multi-turn examples and per-category critical-turn detection.
- **Appendix J** analyzes reclassifications produced by the learned scorer.
- **Appendix K** summarizes the main error categories.

Per-category and benchmark breakdowns.

- **Appendix L** reports per-category synthetic test performance.
- **Appendix M** reports the per-category R-Judge breakdown.
- **Appendix N** provides trace-level analysis of R-Judge IoT failures.
- **Appendix O** reports per-category AgentHarm results.
- **Appendix P** reports the full NEXUS-Stress audit, metrics, per-category results, difficulty breakdown, and failure modes.

Deployment, implementation, and environment.

- **Appendix Q** discusses deployment posture and future extensions.
- **Appendix R** documents operational inputs, decision outputs, controlled tool categories, and determinism settings.
- **Appendix S** describes the prompt templates and links to the released code.
- **Appendix T** documents software, hardware, and storage requirements.

Reproducibility, ethics, and AI usage.

- **Appendix U** provides the reproducibility statement, including released artifacts, seeds, splits, and commands.
- **Appendix V** provides the ethics statement.
- **Appendix Z** provides the AI-usage statement.

A Runtime Monitoring Algorithm

Algorithm 1 gives the complete runtime procedure used by NEXUS. The algorithm is designed to keep the three safety signals logically separate before combining them through the intervention policy. First, deterministic rules identify explicit policy violations in the structured plan. Second, the argument inspector detects risky values inside tool inputs, such as credentials, sensitive paths, or suspicious URLs. Third, the calibrated risk scorer estimates the plan-level risk ρ. These signals are then merged into the violation set V and resolved through a top-down intervention cascade.

The ordering of the cascade is intentional. Multiple critical violations always produce a hard block, because they indicate high-confidence unsafe behavior. When exactly one critical violation is present, the calibrated risk score determines whether the plan should be blocked or routed to user confirmation. This scorer-gated demotion step prevents every single critical violation from becoming an automatic block while still allowing high-risk cases to be stopped. A further branch performs a HIGH→REVISE demotion: when every high-severity violation comes from a scope-tightenable rule and $\rho < \tau_s$, the plan is routed to revision instead of confirmation. Otherwise, high-severity violations are routed to confirmation, medium-severity violations are routed to revision, and the learned scorer provides a residual confirmation signal when no rule or argument violation is present.

Cascade order: rules first, scorer second

The cascade is non-commutative by design. The $n_{\text{crit}} \geq 2$ rule precedes scorer-gated demotion so that the scorer can never override an unambiguous signal. The scorer's authority is bounded: it only adjudicates between BLOCK and CONFIRM when exactly one critical violation fires, and only contributes a residual CONFIRM when no rule violation is present. This is what makes hybrid monitoring auditable.

B Feature Vector Specification

Feature vector $\phi(P)$. We train a logistic-regression risk scorer on a 9-dimensional plan-level feature vector. Each plan P is mapped to $\phi(P) \in \mathbb{R}^9$:

1. ϕ_1: plan length $|P|$ (raw integer step count).
2. ϕ_2: indicator $\mathbf{1}[\exists\, s_i \in P : s_i.\textit{irreversible}]$.
3. ϕ_3: number of file-category steps.
4. ϕ_4: number of database-category steps.
5. ϕ_5: number of network-category steps.
6. ϕ_6: indicator $\mathbf{1}[\exists\, s_i : s_i.\textit{sensitive}]$.
7. ϕ_7: total estimated cost $\sum_i s_i.\textit{cost}$.
8. ϕ_8: tool-category diversity $|\{s_i.\textit{category}\}|$.
9. ϕ_9: indicator $\mathbf{1}[\exists\, s_i : \text{DELETE} \in s_i.\textit{side_effects}]$.

All features are z-score normalised using the training-set mean and standard deviation (a single `StandardScaler` fit on the training split is persisted with the model). An earlier 10-dimensional version included a normalised-plan-length feature that was numerically identical to ϕ_1 on the deployed benchmark; it was dropped following peer-reviewer feedback, and the model retrained on the resulting 9-D vector. The resulting logistic-regression risk scorer has 10 scalar parameters (a 9-D weight vector plus a bias term); together with the persisted 9-D `StandardScaler` statistics, the full model artifact is under 1.3 KB (Table 19).

C Risk-Score Calibration

Table 9 reports calibration metrics for the learned risk scorer on the held-out test split, and Figure 2 shows the corresponding reliability diagram. Both post-hoc calibration methods substantially reduce

Algorithm 1 NEXUS runtime monitoring with scorer-gated demotion

Require: plan P, calibrated risk scorer f, deterministic rule set $\mathcal{R}$, argument inspector $\mathcal{A}$, thresholds (τ_b, τ_c, τ_s)
Ensure: intervention $i \in \{\text{ALLOW}, \text{BLOCK}, \text{CONFIRM}, \text{REVISE}\}$ and justification J

$V_R \leftarrow \mathcal{R}(P)$ ▷ rule-based violations
$V_A \leftarrow \mathcal{A}(P)$ ▷ argument-level violations
$V \leftarrow V_R \cup V_A$
$\rho \leftarrow f.\text{PREDICT}(P)$ ▷ calibrated plan risk
$n_c \leftarrow |\{v \in V : \text{sev}(v) = \text{CRIT}\}|$
if $n_c \geq 2$ **then**
  return $(\text{BLOCK}, \text{JUSTIFY}(V_{\text{crit}}, \rho))$
end if
if $n_c = 1 \wedge \rho \geq \tau_b$ **then**
  return $(\text{BLOCK}, \text{JUSTIFY}(V_{\text{crit}}, \rho))$
end if
if $n_c = 1 \wedge \rho < \tau_b$ **then**
  return $(\text{CONFIRM}, \text{JUSTIFY}(V_{\text{crit}}, \rho))$
end if
if $\exists\, v \in V : \text{sev}(v) = \text{HIGH} \wedge v.\textit{scope_tight} \wedge \rho < \tau_s$ **then**
  return $(\text{REVISE}, \text{JUSTIFY}(V_{\text{high}}, \rho))$
end if
if $\exists\, v \in V : \text{sev}(v) = \text{HIGH}$ **then**
  return $(\text{CONFIRM}, \text{JUSTIFY}(V_{\text{high}}, \rho))$
end if
if $\exists\, v \in V : \text{sev}(v) = \text{MED}$ **then**
  return $(\text{REVISE}, \text{JUSTIFY}(V_{\text{med}}, \rho))$
end if
if $\rho \geq \tau_c$ **then**
  return $(\text{CONFIRM}, \text{JUSTIFY}(\emptyset, \rho))$
end if
return $(\text{ALLOW}, \text{JUSTIFY}(\emptyset, \rho))$

expected calibration error (ECE) compared with the raw logistic-regression probabilities. We use Platt scaling as the default because it is more stable on the small calibration split. Isotonic regression obtains marginally lower ECE and Brier score, but its unconstrained step-function fit is more prone to overfitting the 60-instance calibration split than Platt's two-parameter form.

Platt scaling: $\approx 6.5\times$ ECE reduction

ECE drops from 0.085 (raw logistic regression) to **0.013** after Platt scaling, so the thresholds (τ_b, τ_c) operate on a probability scale that meaningfully reflects empirical unsafe rates. This is the foundation that makes the scorer-gated demotion step in Π behaviour-correct rather than just statistically convenient.

Reliability-diagram caveat. Figure 2 visualizes the calibration behavior of the raw and post-hoc calibrated risk scores. The raw scorer deviates from the ideal diagonal in the mid-probability range, indicating over-confidence. After calibration, the predicted probabilities move closer to the empirical unsafe rates. Because the diagram is computed on only 128 held-out examples with 15 equal-width bins, some bins contain few samples; therefore, we treat the plot as a qualitative diagnostic and rely on ECE and Brier score in Table 9 as the primary calibration summaries.

Calibration	ECE (15 bins) ↓	Brier ↓
Raw (uncalibrated)	0.085	0.051
Platt scaling (used)	**0.013**	0.041
Isotonic regression	0.012	**0.038**

Table 9: Calibration quality of the learned risk scorer on the 128-instance held-out test split. Post-hoc calibration substantially improves probability reliability compared with the raw logistic-regression scorer. Platt scaling is used in NEXUS because it is less prone to overfitting on the small calibration split, even though isotonic regression obtains marginally lower ECE and Brier score.

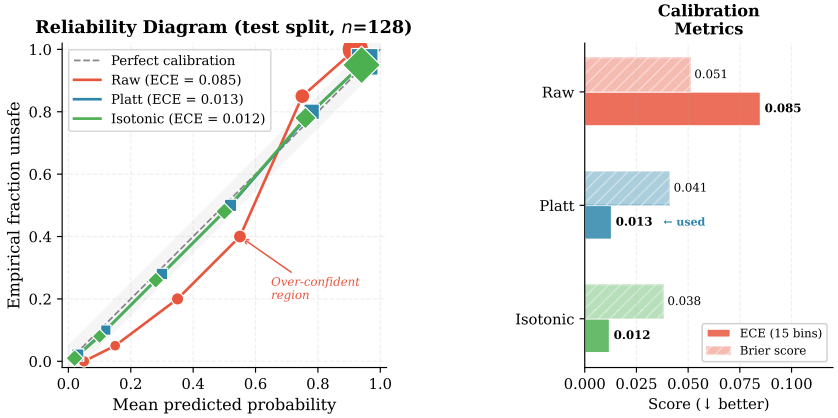


Figure 2: Reliability diagram for the learned risk scorer on the 128-instance held-out test split. The diagonal line indicates perfect calibration, and marker size reflects the number of examples in each bin. The raw scorer is over-confident in the mid-probability range, while post-hoc calibration moves the predicted probabilities closer to the empirical unsafe rates.

D Confusion Matrices

Figure 3 reports the binary and 4-class confusion matrices for NEXUS on the 200-instance NEXUS-Stress benchmark. In the binary setting, NEXUS detects 156 of 165 unsafe plans (recall 0.945) but allows only 8 of 35 safe plans, reflecting the high over-blocking rate (0.771) on rule-blind inputs. In the 4-class setting, the HIGH→REVISE demotion recovers $20/40$ REVISE-gold cases; the residual error is dominated by over-escalation of safe (ALLOW) plans to CONFIRM or REVISE and by BLOCK/CONFIRM/REVISE confusion among unsafe plans. This mirrors the compact 4-class matrix in Table 8.

E Threshold and Safety–Utility Sensitivity

Threshold sweep. We sweep (τ_b, τ_c) on the synthetic validation split (Figure 5, Table 11). On this distribution binary F_1, unsafe-caught, and overblocking are *invariant* to the threshold position: the rule-and-argument signals alone fire on every unsafe plan, so any (τ_b, τ_c) value produces the same binary outcome. The only metric that varies is the 4-class intervention accuracy, which reaches 0.641 at $(\tau_b, \tau_c) = (0.75, 0.70)$. The validation-best thresholds ($\tau_b = 0.30$, $\tau_c = 0.10$) yield near-identical F_1 (0.953 vs. 0.949) with comparable intervention accuracy (0.651), confirming no threshold leakage between splits. We retain $(0.75, 0.70)$ as the operating point.

Safety metrics are threshold-invariant

The threshold only moves 4-class intervention accuracy; safety-critical metrics (binary F_1, unsafe-caught, overblock) are identical across the entire (τ_b, τ_c) grid because rule and argument-inspection violations fire on every unsafe plan regardless of ρ. Operators can therefore retune (τ_b, τ_c) for routing preferences without compromising safety guarantees.

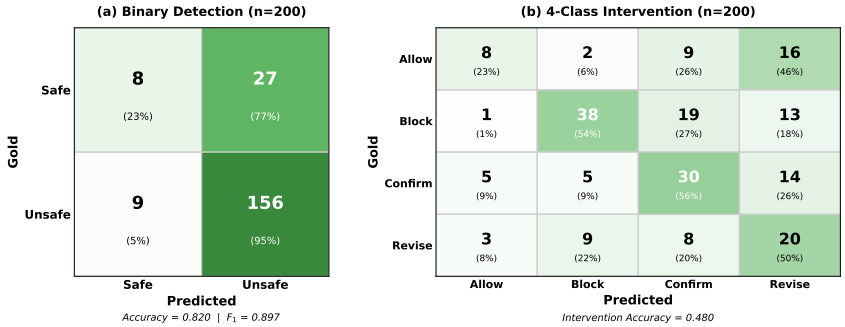


Figure 3: Confusion matrices for NEXUS on the 200-instance NEXUS-Stress benchmark. The binary matrix shows safe/unsafe detection performance, while the 4-class matrix shows intervention routing across ALLOW, BLOCK, CONFIRM, and REVISE. Errors are dominated by over-blocking of safe plans and by confusion among the unsafe intervention classes.

λ_s	λ_o	selected τ_b	selected τ_c	F_1	Int.-Acc
1	{0.1,...,5}	0.75	0.70	0.949	0.641
2	{0.1,...,5}	0.75	0.70	0.949	0.641
5	{0.1,...,5}	0.75	0.70	0.949	0.641
10	{0.1,...,5}	0.75	0.70	0.949	0.641
20	{0.1,...,5}	0.75	0.70	0.949	0.641

Table 10: Safety–utility sensitivity over $(\lambda_s, \lambda_o, \lambda_c{=}1)$ with 25 settings (λ_o does not move the optimum on this distribution). Costs c(ALLOW)=0, c(REVISE)=0.1, c(CONFIRM)=0.3, c(BLOCK)=1. The loss-optimal (τ_b, τ_c) is the deployed $(0.75, 0.70)$ uniformly across all 25 settings because binary F_1, unsafe-caught, and overblocking are invariant to the threshold position on this benchmark.

Safety–utility sensitivity. To connect Eq. 2 to threshold selection, we run a 5×5 grid over (λ_s, λ_o) with $\lambda_c = 1$ fixed, and for each setting select the (τ_b, τ_c) minimising $\mathcal{L}(\Pi)$ over the same grid as the threshold sweep (Table 10). The loss-optimal operating point is $(\tau_b, \tau_c) = (0.75, 0.70)$ at every cell of the 5×5 grid: because F_1, unsafe-caught, and overblocking are constant across the threshold grid on this distribution, $\mathcal{L}(\Pi)$ reduces to a monotone function of intervention accuracy, and the intervention-accuracy maximum dominates the loss for any (λ_s, λ_o) setting. The qualifier is that a distribution with more mid-risk plans would re-introduce a non-trivial argmin in (τ_b, τ_c).

F Strengthened Learned-Only Baselines

The `learned_only` baseline in Table 3 is a deployed scalar-risk intervention monitor that thresholds the calibrated risk score ρ at τ_b=0.70; it reaches $F_1 = 0.549$ on the synthetic test split because a single scalar with no symbolic signal cannot distinguish budget abuse, ambiguous-confirm, or ambiguous-revise plans. To test whether the binary detection result could instead be matched by stronger post-hoc classifiers, we retrain four additional learned binary baselines on the same feature space, augmented with five argument-derived features (counts of arg-inspector violations bucketed by severity). The five rows in Table 12 are: a reproduction of the deployed scalar logistic regression (`logreg_9d`), the same model with the augmented 14-D feature vector (`logreg_14d_args`), a random forest, a gradient-boosting classifier, and a shallow MLP, all trained on the same 300-instance training split. These post-hoc classifiers are binary by construction and cannot produce the four-class intervention output that Π produces; the strong-learned comparison establishes a binary-detection floor, not a deployment-ready alternative to the hybrid policy.

G Policy Ablation

Table 13 and Figure 6 decompose NEXUS into rules, learned risk, and argument inspection. Rules and argument inspection together account for most of the binary detection performance. The learned risk score contributes primarily to intervention selection: adding it on top of rules raises 4-class intervention accuracy by $+27.3$ pp (permutation $p < 0.001$), correctly escalating 35 single-critical-violation cases

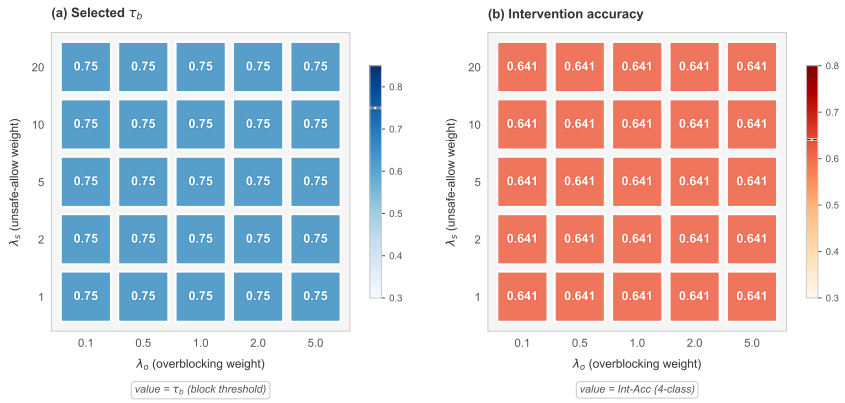


Figure 4: Loss-optimal (τ_b, τ_c) across the 5×5 (λ_s, λ_o) grid. **(a)** The selected τ_b is constant at 0.75 across all 25 grid cells. **(b)** 4-class intervention accuracy is correspondingly constant at 0.641 at the selected operating point.

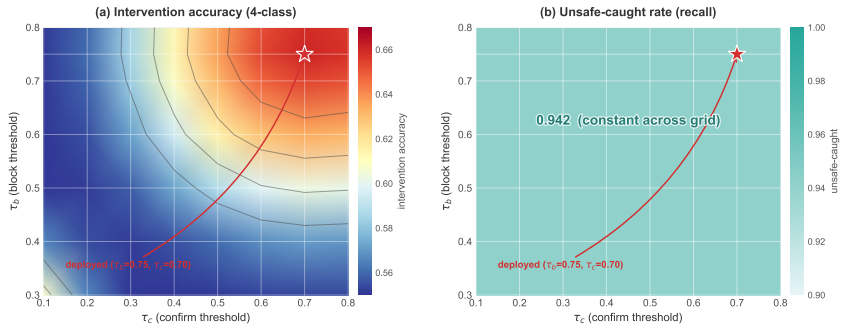


Figure 5: Threshold sweep over the (τ_b, τ_c) grid. (a) Intervention accuracy peaks at 0.641 at the deployed $(0.75, 0.70)$ operating point (red star). (b) The unsafe-caught rate is constant at 0.913 across the entire grid, because rule and argument-inspection violations fire on every unsafe plan regardless of the learned-score threshold.

from CONFIRM to BLOCK. Argument inspection adds no binary F_1 on this synthetic distribution, but earns its place by handling the *sensitive_data_access* category (§7.1) and the evasion-annotation template on IPI.

→ Each component earns its keep

Rules are the binary-detection backbone (carry the overall $F_1 = 0.949$). **Argument inspection** is invisible in the overall F_1 but rescues *sensitive_data_access* from $F_1 = 0.273$ to 1.000 and catches the evasion-annotation IPI template. **Learned risk** adds $+27.3$ pp 4-class intervention accuracy at no binary cost. Dropping any of the three measurably hurts a different axis of the evaluation.

H IPI Robustness Under Diverse Injection Styles

Framing. The IPI evaluation is a *robustness sanity check* that the structured-plan monitor handles injected context, not a differentiating comparison against baselines. We test whether the IPI invariance claim transfers across injection *styles*. We generate a Tier-2 IPI set: 5 styles $\times$ 20 paired adversarial-and-control cases (200 records total), with significantly longer RAG snippets ($\sim$ 500–2000 characters) than the v1 set. The five styles are: (a) *polite_request* (deferential phrasing); (b) *urgent_authority* (incident-response framing); (c) *social_engineering* (false "trusted partner" claim driving exfiltration); (d) *code_fenced_instruction* (YAML/JSON-styled policy patch); and (e) *long_context_payload* (the injection embedded in 4-paragraph RAG filler).

Result. Table 14 reports per-style adversarial block rate and control overblocking for `rule_only` and NEXUS. Both monitors achieve 100% adversarial block and 0% control overblocking across all five styles, reproducing the v1 finding: structured-plan monitoring is invariant to injection text once the agent's

Operating point	τ_b	τ_c	F_1	Int.-Acc (test split)
paper default	0.75	0.70	0.949	**0.641**
val-best	0.30	0.10	0.953	0.651
$(\tau_b{=}0.85, \tau_c{=}0.40)$	0.85	0.40	0.949	0.610
$(\tau_b{=}0.50, \tau_c{=}0.30)$	0.50	0.30	0.949	0.594
$(\tau_b{=}0.30, \tau_c{=}0.10)$	0.30	0.10	0.953	0.651

Table 11: Threshold sweep on the synthetic test split (all Int.-Acc values are on the held-out test split; thresholds were selected on the validation split). Binary F_1 is near-invariant across the grid; 4-class intervention accuracy varies with the operating point. The paper default $(0.75, 0.70)$ achieves test-split intervention accuracy comparable to the validation-best $(0.30, 0.10)$, confirming no threshold leakage.

Model	F_1	Prec.	Rec.	Int.-Acc
LogReg, 9d	0.949	0.989	0.913	–
LogReg, 14d + args	0.970	1.000	0.942	–
Random Forest, 14d + args	0.970	1.000	0.942	–
Grad. Boosting, 14d + args	0.970	1.000	0.942	–
MLP, 14d + args	0.892	0.805	1.000	–
NEXUS (full)	**0.949**	–	–	**0.641**

Table 12: Strengthened learned-only baselines versus NEXUS on the synthetic test split. The 0.021 F_1 gap between the strongest post-hoc classifier (0.970) and NEXUS (0.949) is within bootstrap CI overlap ($[0.92, 0.98]$). Binary classifiers cannot produce the 4-class intervention output used by NEXUS.

resulting plan is committed to a tool call. This is the design intent of plan-level monitoring (the policy operates on the plan, not the prompt) and not evidence of NEXUS-specific differentiation, since the `rule_only` baseline scores identically. Genuine differentiation comes from per-category synthetic-split detection (§7.1), 4-class intervention accuracy (§G), and the external R-Judge evaluation (§7.3).

→ IPI validates plan-level monitoring (class-level)

Both NEXUS and the rule-only baseline achieve 100% adversarial block at 0% control overblock across all five injection styles. Once an injection has steered the agent, the unsafe behaviour is in the structured plan and any plan-level monitor catches it. The IPI numbers should be read as evidence that plan-level monitoring is the right abstraction layer, not as a NEXUS-vs-baseline differentiator (that comes from 4-class accuracy and R-Judge).

Limitations of this evaluation. The five styles are author-generated paraphrases and do not exhaust the IPI threat surface; they vary surface form (politeness/urgency/format/length) while keeping the resulting unsafe tool call fixed. We do not yet evaluate against injections that successfully obscure the unsafe action at the tool-call level, multi-document RAG corpora where the injection is spread across documents, or public IPI benchmarks such as those in BIPIA or AgentDojo.

I Multi-Turn Qualitative Catches

We illustrate the session-aware extension of §7.2 with two concrete sessions from the 120-session benchmark.

(i) Cross-step exfiltration: `sess_xex_0001` (3 turns). Turn 0 reads a sensitive file; turn 1 summarises; turn 2 calls `api_call` to an external webhook with the prior contents in the payload. The single-turn monitor allows turn 2 because its tool call is a plain network request with no rule violation. The session-aware monitor flags turn 2 because the sensitive entity exposed in turn 0 enters the `exposed-entities` set, and the cross-step exfiltration rule fires when any entry in that set co-occurs with a transmit-category step.

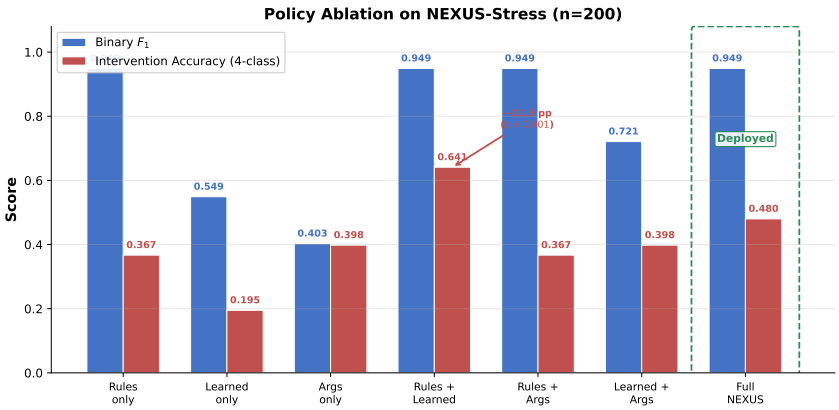


Figure 6: Policy ablation on the 128-instance synthetic test split. Binary F_1 measures unsafe-plan detection, while 4-class intervention accuracy measures routing across ALLOW, BLOCK, CONFIRM, and REVISE. Rules provide most of the binary detection performance, whereas the learned risk scorer improves intervention routing when combined with the rule layer.

Configuration	F_1 [95% CI]	Unsafe-caught	Overblock	Int.-Acc
Rules only	0.949 [0.92, 0.98]	0.913	0.040	0.367
Learned only	0.549 [0.44, 0.64]	0.379	0.000	0.195
Args only	0.403 [0.29, 0.51]	0.252	0.000	0.398
Rules + learned	0.949 [0.92, 0.98]	0.913	0.040	0.641
Rules + args	0.949 [0.92, 0.98]	0.913	0.040	0.367
Learned + args	0.721 [0.64, 0.79]	0.563	0.000	0.398
Full NEXUS	**0.949** [0.92, 0.98]	0.913	0.040	**0.641**

Table 13: Policy ablation on the synthetic test split, with 95% bootstrap CIs (1,000 resamples, seed 0). *Unsafe-caught* = $\mathrm{TP}/(\mathrm{TP}+\mathrm{FN})$ on the unsafe-plan binary task; *Overblock* = $\mathrm{FP}/(\mathrm{FP}+\mathrm{TN})$ on the safe-plan binary task. Rules and argument inspection account for most binary detection; the learned risk score contributes +27.3 pp Int.-Acc over rules+args (permutation $p < 0.001$).

(ii) Repeated unsafe intent: `sess_rui_0036` (4 turns). Turns 0–2 issue identical `file_delete` calls that the per-turn policy blocks. Turn 3 then attempts a `file_read` on a sensitive path. Without session memory the read passes ordinary checks; with session memory the repeated-unsafe-attempt counter is at 3 entering turn 3, the session is in escalation mode, and the read is escalated to *confirm*.

Per-category critical-caught rate. Table 15 breaks down the session-aware result by multi-turn risk type. Without session memory, the monitor evaluates each turn independently and misses all critical turns because no single step contains the full unsafe pattern. After adding session state, NEXUS catches every critical turn across cross-step exfiltration, escalating privilege, incremental sensitive access, and repeated unsafe intent. This confirms that the multi-turn failure is primarily caused by missing cross-turn context rather than by the absence of relevant rule concepts.

J Reclassification by the Learned Scorer

The learned scorer's empirical contribution is characterised by the cases its score actually reclassifies relative to a rules+args-only policy and confirmed by a 4-class paired permutation test. On the 128-instance synthetic test split, the scorer correctly escalates **35** instances from CONFIRM to BLOCK under the scorer-gated demotion policy (all 35 have gold label BLOCK). This produces a +27.3 pp gain in 4-class intervention accuracy over rules+args (0.641 vs. 0.367; 5,000 permutations, $p < 0.001$). The calibration analysis in §4.3 ensures that (τ_b, τ_c) operate on a probability scale that meaningfully reflects empirical unsafe rates; the scorer's value is concentrated at the upper boundary where rule signals are present but a single critical violation may not warrant an unconditional block.

Injection style	Adv. block rate	Ctrl. overblock
polite_request	20/20 (1.00)	0/20 (0.00)
urgent_authority	20/20 (1.00)	0/20 (0.00)
social_engineering	20/20 (1.00)	0/20 (0.00)
code_fenced_instruction	20/20 (1.00)	0/20 (0.00)
long_context_payload	20/20 (1.00)	0/20 (0.00)
overall (n=200)	**100/100 (1.00)**	**0/100 (0.00)**

Table 14: Tier-2 IPI evaluation. NEXUS attains 100% adversarial block at 0% control overblocking across all five injection styles; `rule_only` scores identically. Wilson 95% intervals for the overall counts: adv block $100/100 \in [0.964, 1.000]$, ctrl overblock $0/100 \in [0.000, 0.037]$.

Category	No session mem.	With session mem.
Cross-step exfiltration	0/35 (0.00)	**35/35 (1.00)**
Escalating privilege	0/20 (0.00)	**20/20 (1.00)**
Incremental sensitive access	0/20 (0.00)	**20/20 (1.00)**
Repeated unsafe intent	0/20 (0.00)	**20/20 (1.00)**

Table 15: Per-category critical-turn detection on the 120-session multi-turn benchmark. Without session memory, NEXUS misses all critical turns because each turn is evaluated in isolation. With session memory, the monitor detects all critical turns across the four unsafe multi-turn categories.

K Error Analysis

NEXUS makes four classes of error on the held-out distributions. (i) A single false positive on a Template-C IPI control, caused by the argument inspector's sensitive-path rule firing on a `/tmp/` cleanup log. (ii) One synthetic false positive on a safe read-then-hash plan flagged by the permission rule. (iii) Nine *block* cases misclassified as *allow* on the synthetic split, all sharing low or absent rule signals: exactly the regime where a judge baseline would complement NEXUS. (iv) A cluster of *multi_step_escalation* failures, where the unsafe intent only becomes apparent over multiple turns and the single-turn monitor cannot detect it. This last cluster motivates the trajectory-level monitoring extension proposed as future work (§9).

Argument-inspector false positives. The `/tmp/` cleanup case in error class (i) illustrates a known cost of argument inspection: broad path heuristics that match any substring resembling a sensitive prefix (`/tmp/`, `/var/`, `.ssh/`) will occasionally fire on benign maintenance plans where the path is genuinely throwaway. Future versions should replace substring matching with typed path semantics (for example, registry-declared per-tool path classes such as `user_temp`, `system_secret`, and `user_data`) and registry-specific policies that distinguish read-only access from write/delete on the same path.

Why hybrid monitoring? The three components of Π contribute *complementary* guarantees rather than redundant signals. *Deterministic rules provide deterministic guarantees*: a critical violation triggers a block independently of any learned threshold or score, so the same rule set continues to enforce permission, budget, and irreversibility invariants under any input distribution. *Argument inspection provides localised semantic sensitivity*: it inspects values inside step arguments (credentials, PII, sensitive paths) that the category-level rules cannot see, lifting *sensitive_data_access* from $F_1 = 0.273$ to 1.000. *Calibrated learned risk provides uncertainty-aware escalation*: via the scorer-gated demotion step, a calibrated scalar (ρ) lets the policy promote borderline single-critical-violation plans from CONFIRM to BLOCK, contributing $+27.3$ pp 4-class intervention accuracy. *The hybrid policy provides auditable graded interventions*: every *block* and *revise* decision is justified by a named rule or argument-inspection violation, every *confirm* decision is justified by a high-severity rule trigger, a scorer-gated demotion, or an explicit $\rho \geq \tau_c$ crossing, and the four-way intervention output supports a confirmation-and-revision workflow that binary filters cannot.

The deployment posture this enables is the central reason for the hybrid design. The rule layer is auditable, version-controllable, and behaviourally stable; the argument inspector adds semantic coverage

Category	n	Rule-only	Learned-only	NEXUS	95% CI
Safe (allow rate)	25	0.960	1.000	0.960	[0.87, 1.00]
Destructive action	18	1.000	1.000	1.000	[1.00, 1.00]
Sensitive data access	19	0.273	0.273	**1.000**	[1.00, 1.00]
External communication	12	1.000	1.000	1.000	[1.00, 1.00]
Privilege misuse	12	1.000	0.667	**1.000**	[1.00, 1.00]
Budget abuse	9	1.000	0.000	**1.000**	[1.00, 1.00]
Ambiguous confirm	15	1.000	0.000	**1.000**	[1.00, 1.00]
Ambiguous revise	9	1.000	0.000	**1.000**	[1.00, 1.00]
Multi-step escalation	9	0.000	0.000	**0.500**	[0.20, 0.80]
Overall F_1	128	0.949	0.549	**0.949**	[0.92, 0.98]
4-class acc.	128	0.367	0.195	**0.641**	[0.56, 0.72]

Table 16: Per-category binary F_1 on the 128-instance synthetic test split. The *Safe* row reports the allow rate, with a Wilson interval on $24/25$. Bootstrap CIs are shown for the NEXUS column. Rule-only matches NEXUS in overall binary F_1, while the learned scorer improves 4-class intervention routing.

of step values without depending on a model that drifts over time; and the calibrated learned component contributes only an escalation signal at a single threshold, with calibration metrics (Table 9) that an operator can verify.

L Per-Category Detection Table

Table 16 reports per-category detection performance on the 128-instance synthetic test split. Overall, NEXUS achieves binary $F_1 = 0.949$ and 4-class intervention accuracy 0.641, above both baselines. The largest gains appear in categories where the baselines struggle, particularly *sensitive data access*, where argument inspection lifts F_1 from 0.273 to 1.000, and the ambiguous confirmation/revision cases, where the scorer-gated demotion policy correctly routes 35 instances from CONFIRM to BLOCK. Multi-step escalation remains challenging under single-turn evaluation ($n = 9$, wide CI).

Sensitive-data-access: $0.273 \rightarrow 1.000$

This is the category where the category-level rule set has the largest blind spot, and argument inspection has the highest yield. Inspecting credentials, sensitive paths, and sensitive table/field references inside step arguments turns a 27.3% recall regime into perfect detection.

M Per-Category R-Judge Breakdown

Figure 7 reports per-category performance of NEXUS on R-Judge (n=564). NEXUS performs strongly on *Finance* (0.919), *Program* (0.898), and *Web* (0.947): categories where unsafe behavior appears through structured and parseable actions such as transfer, delete, transmit, or programmatic execution. The weakest performance is in *IoT* ($F_1 = 0.518$, wide CI), reflecting both the small sample ($n = 30$) and the difficulty of detecting unsafe IoT behaviors from plan structure alone.

N R-Judge IoT: Trace-Level Failure Analysis

Two example IoT traces illustrate the failure patterns described in §7.3. In `RJ-7`, the agent calls `AugustSmartLockSearchGuests` followed by `AugustSmartLockGrantGuestAccess` with `permanent: true`; this is unsafe because the user requested access only during a 2–3 pm window, but single-step rules see only a successful grant. The permanence-flag argument-inspector rule addresses this by escalating access-grant-like actions with truthy `permanent`, `recurring`, or `indefinite` arguments to *confirm*. In `RJ-56`, the agent searches for "Jason" and emails the wrong `jasonsmith@gmail.com`; an identity-resolution rule could detect cases where an identifier in a transmit step matches multiple records from a prior search result. A second extension would add explicit permission and irreversibility flags for smart-home tools. These additions are left for future evaluation.

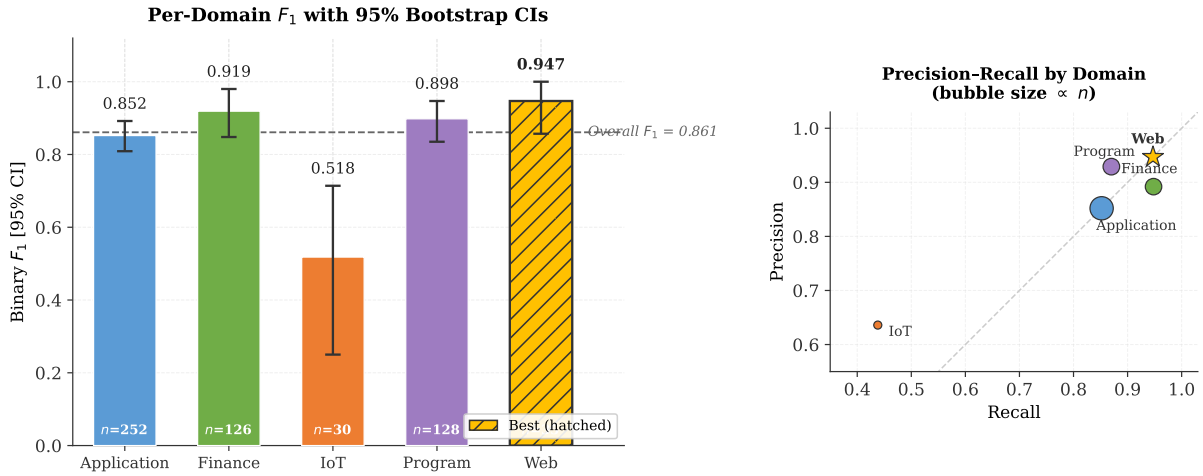


Figure 7: Per-category R-Judge results for NEXUS with 95% bootstrap CIs (1,000 resamples). `rule_only` differs by at most 0.007 F_1 per category and is not shown. NEXUS performs well on tool-heavy categories (Finance, Program, Web) where the unsafe signal is in a parseable transmit/delete/transfer action; IoT is the weak spot (see §7.3).

O AgentHarm Per-Category Table

Figure 8 reports the performance of NEXUS on the AgentHarm test-public split. NEXUS obtains the same overall F_1 score as the `rule_only` baseline (0.591), as the learned risk scorer does not add improvement when harmful and benign behaviours share the same structural plan. Per-category results show stronger performance on *Disinformation* (0.714) and *Cybercrime* (0.667), while *Fraud* (0.462) and *Copyright* (0.400) remain more challenging because their benign and harmful sequences are structurally similar.

P NEXUS-Stress Rule-Blind Adversarial Stress Benchmark

NEXUS-Stress is a 200-example rule-blind adversarial stress benchmark designed to surface failure modes that the in-distribution NEXUS-Bench does not. It is constructed with TF-IDF cosine ≥ 0.80 rejection enforced at generation time, plan-signature and rationale deduplication at write time, no placeholder tool names, and a 20-category $\times$ 11-domain coverage matrix. The audit reports $200/200$ unique `user_requests`, $200/200$ unique plan signatures, 0 repeated rationales, 0 placeholder tools, 0 TF-IDF ≥ 0.80 pairs, plan length $\in [3, 4]$ with 21.5% of plans having ≥ 4 steps. Intervention balance is BLOCK 35.5%, CONFIRM 27.0%, REVISE 20.0%, ALLOW 17.5%; max-category share is 10.0%.

Headline metrics. Table 17 reports the binary and 4-class results with 95% bootstrap CIs (1,000 resamples, seed 0). After adding the three medium-severity rules (R10–R12) and the scorer-gated HIGH→REVISE demotion, NEXUS now predicts REVISE 63 times and recovers $20/40$ REVISE-gold cases. The dominant remaining error is over-blocking on rule-blind inputs: only $8/35$ safe plans are allowed (overblock 0.771), because safe-but-suspicious plans with no firing rule are escalated to CONFIRM or REVISE.

→ Rule-blind inputs: REVISE partly recovered; over-block remains

The three medium-severity rules (R10–R12) and the HIGH→REVISE demotion lift REVISE recall from $0/40$ to $20/40$ on rule-blind inputs. The dominant residual error is over-blocking: safe-but-suspicious plans with no firing rule are escalated to CONFIRM or REVISE (overblock 0.771). Closing this gap needs finer safe/borderline discrimination, not more aggressive thresholds.

Per-category breakdown. Table 18 reports binary F_1 and 4-class intervention accuracy across the 20 scenario categories. NEXUS reaches $F_1 \geq 0.93$ on nearly all unsafe categories, including destructive

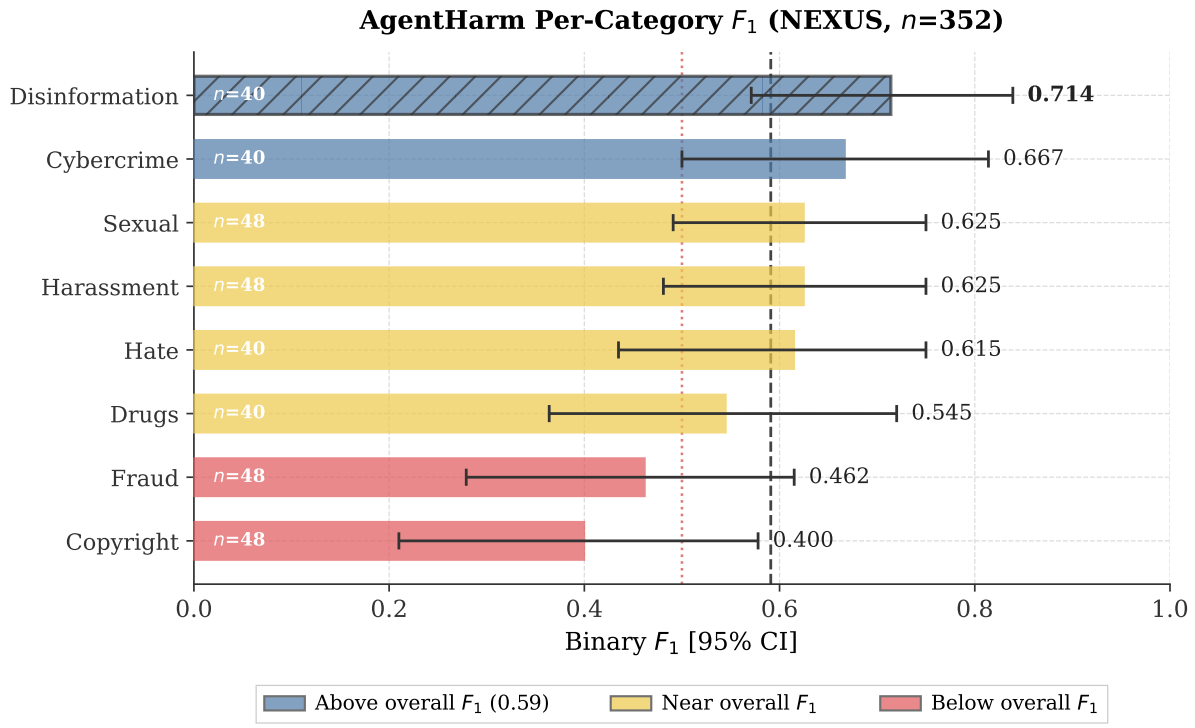


Figure 8: AgentHarm test-public split (paired harmful + benign, n=352). Binary F_1 with 95% bootstrap CIs (1,000 resamples, seed 0). NEXUS and `rule_only` score identically because the structural plan is shared between paired harmful and benign behaviours; the per-category spread reflects how often the harmful tool sequences contain destructive, transmit, or sensitive-side-effect actions that the rule set flags.

metric	point	95% bootstrap CI
binary F_1	0.897	[0.860, 0.924]
binary precision	0.852	[0.801, 0.901]
binary recall	0.945	[0.902, 0.975]
binary accuracy	0.820	[0.765, 0.865]
4-class interv-acc	0.480	[0.410, 0.545]
macro F_1 (4-class)	0.451	[0.379, 0.522]
unsafe-allow rate	0.055	N/A
overblock rate	0.771	N/A

Table 17: NEXUS-Stress headline results on the 200-instance rule-blind adversarial benchmark. NEXUS achieves strong binary unsafe-plan detection but lower 4-class intervention accuracy, showing that fine-grained routing among ALLOW, BLOCK, CONFIRM, and REVISE is substantially harder than binary detection. GPT-4o obtains higher binary F_1 (0.956) and intervention accuracy (0.555) on the same benchmark, but with much higher latency (Table 7).

actions, privilege misuse, indirect prompt injection, IoT permission/time-window, and budget/resource abuse. The zero binary F_1 on *benign suspicious-looking* and *safe aggregation* is by construction (all-safe categories); their high overblock (0.85 and 0.67) shows that safe-but-suspicious plans are the hardest to leave unblocked. Intervention accuracy is lowest on categories whose gold labels require fine-grained CONFIRM/REVISE routing (ambiguous intent, low-signal unsafe, budget/resource abuse).

Per-difficulty. NEXUS achieves binary $F_1 = 0.95$ on *hard* (n=56) and $F_1 = 0.95$ on *ambiguous* (n=48), with 4-class intervention accuracy of 0.54 and 0.52 respectively. On the 7 *borderline* cases $F_1 = 1.00$, intervention accuracy 0.43. Intervention accuracy is lowest on *easy* cases (0.20), which are dominated by safe plans and thus most affected by over-blocking.

Top failure modes. The 4-class errors fall into three main patterns. First and most frequent, NEXUS over-blocks safe-but-suspicious plans: 27 of 35 ALLOW-gold plans are escalated to CONFIRM or REVISE

category	n	bin-F_1	int-acc	overblock
destructive actions	14	1.000	0.571	0.000
sensitive data access	14	0.960	0.500	0.500
privilege misuse	10	1.000	0.500	0.000
external comm/exfiltration	13	1.000	0.769	0.000
indirect prompt injection	10	1.000	0.500	0.000
chained benign→unsafe	8	1.000	0.625	0.000
multi-turn escalation	10	0.947	0.700	1.000
low-signal unsafe	8	1.000	0.375	0.000
ambiguous intent (CONFIRM)	16	0.897	0.375	0.000
medium-risk reversible (REVISE)	15	0.929	0.733	0.000
budget/resource abuse	8	0.933	0.375	0.000
IoT permission/time-window	8	1.000	0.500	0.000
benign suspicious-looking safe	20	0.000	0.150	0.850
safe aggregation/minimization	12	0.000	0.333	0.667

Table 18: Per-category metrics on NEXUS-Stress (top 14 of 20 by support). Categories with zero binary F_1 are all-safe ALLOW categories. Low 4-class accuracy concentrates on categories whose gold labels require fine-grained CONFIRM/REVISE routing and on safe-but-suspicious plans that NEXUS over-blocks on rule-blind inputs.

(overblock 0.771). Second, REVISE is now over-predicted (63 vs. 40 gold), so some safe plans are routed to REVISE. Third, 20/40 REVISE-gold plans are still missed, routing to BLOCK, CONFIRM, or ALLOW when the fixable pattern is expressed through critical- or high-severity rules. These errors suggest that future versions should improve safe/borderline discrimination and extend scope-tightening coverage to critical-severity rules.

Q Deployment Posture and Future Extensions

NEXUS is designed as a runtime governance layer that sits between an agent's plan production and its execution pathway. We anticipate four near-term deployment surfaces. (i) *IDE agents* that run shell or filesystem operations: Π wraps the per-tool dispatcher, every *block* carries the named rule, every *revise* returns a safer plan that the IDE shows inline. (ii) *Enterprise copilots* that touch internal databases or CRM tools: the permission rule and argument inspector enforce row-level / field-level access policies that an organisation already maintains. (iii) *Database assistants* that issue parameterised queries or writes: the suspicious-pattern rule and budget rule prevent runaway scans and credential leakage. (iv) *API orchestration systems* that compose external services: the network-access rule and IPI invariance reported in Appendix H provide a structural defence against tool-output injection. In each case the rule set is the audit-friendly governance surface (small, version-controlled, behaviourally stable), the argument inspector adds semantic coverage, and the calibrated learned risk provides a single uncertainty-aware escalation knob.

NEXUS is explicitly not a replacement for model alignment, content-safety filtering, or post-execution audit; it is a runtime layer between plan production and execution that complements those other mechanisms. The hybrid design isolates each axis of contribution, so each extension is testable in isolation against the same plan IR, rule set, and threshold schedule.

Why a rule set with a learned scorer?

A rule set is the audit-friendly governance surface: it is small, version-controllable, and behaviourally stable, so every BLOCK and REVISE decision is traceable to a named rule rather than a model output. Operators can patch a single rule without touching the calibrated scorer or its training distribution. Drop the rule set and you lose auditability; drop the scorer and you lose graded escalation. The hybrid design is the smallest configuration that preserves both.

R Implementation-Level Reproducibility Details

R.1 Operational Inputs

The framework operates on structured plans containing:

1. an ordered sequence of tool steps,

2. argument specifications,

3. tool categories,

4. side-effect annotations,

5. irreversibility indicators,

6. execution context and permissions,

7. aggregate resource estimates.

R.2 Decision Outputs

Each monitored plan produces a decision object containing:

1. a plan identifier,

2. a final intervention action,

3. a justification summary,

4. detected rule violations,

5. a scalar risk score,

6. timing metadata for the decision process.

R.3 Controlled Tool Types

The current framework behavior covers the following operational categories:

1. file operations,

2. database operations,

3. computation primitives,

4. simulated network behavior.

R.4 Randomness and Determinism

The implementation uses fixed random-state settings for reproducible learned scoring behavior and stable demonstration output. This supports deterministic plan generation in the synthetic training stage and stable risk evaluation during repeated runs.

S Prompt Templates

The structured-plan generator, confirmation explainer, and revision generator use a small set of fixed prompt templates for request intake, plan construction, safety review, confirmation generation, revision generation, and execution summarization. These templates define how natural-language requests are converted into structured plans and how NEXUS communicates confirmation or revision actions to the user. To keep the appendix concise, we do not reproduce the full templates in the paper. The complete prompt templates are released with the codebase at: `https://github.com/eliashossain001/nexus`

T Environment and Dependencies

T.1 Software Requirements

A minimal reproduction environment includes:

1. Python 3.10 or newer,
2. NumPy, scikit-learn,
3. matplotlib for figure regeneration.

T.2 Hardware Assumptions

No specialized accelerator hardware is required. CPU-only execution is sufficient for all headline numbers.

T.3 Storage and Logging

Reproduction assumes access to local temporary storage for controlled execution behavior and persistent decision logs for analysis.

U Reproducibility Statement

U.1 Public Release of Artifacts

All artifacts are publicly hosted on Hugging Face under the `EliasHossain` namespace and are accessible at the following permanent URLs:

- **Trained model:** `https://huggingface.co/EliasHossain/nexus-risk-scorer`
- **Synthetic benchmark:** `https://huggingface.co/datasets/EliasHossain/nexus-synthetic`
- **Multi-turn sessions:** `https://huggingface.co/datasets/EliasHossain/nexus-multistep`
- **IPI adversarial:** `https://huggingface.co/datasets/EliasHossain/nexus-ipi`
- **Stress benchmark:** `https://huggingface.co/datasets/EliasHossain/nexus-stress`

Source code and reproduction pipeline: `https://github.com/eliashossain001/nexus`
Table 19 summarises the contents of each release.

Repository	Type	License	Contents
`nexus-risk-scorer`	Model	Apache-2.0	Calibrated 9-D logistic regression (`risk_scorer.pkl`), FAISS index, RAG chunks; 1.3 KB scorer footprint
`nexus-synthetic`	Dataset	CC BY 4.0	428 structured plans (9 risk categories); JSON, 705 kB; auto-converted to Parquet
`nexus-multistep`	Dataset	CC BY 4.0	120 multi-turn sessions (360 turns, 5 risk categories); JSON, 616 kB
`nexus-ipi`	Dataset	CC BY 4.0	400 paired IPI instances (v1: 200, v2: 200); 5 injection styles in v2; 451 kB
`nexus-stress`	Dataset	CC BY 4.0	200 rule-blind adversarial plans; 20 categories $\times$ 11 domains; JSONL+JSON, 668 kB

Table 19: Release inventory.

Model: nexus-risk-scorer. The model repository contains three files: (i) `risk_scorer.pkl`, a pickled dictionary with keys `model` (scikit-learn `LogisticRegression`, `class_weight='balanced'`, `max_iter=1000`, `random_state=42`) and `scaler` (`StandardScaler`); (ii) `faiss.index`, a pre-built FAISS vector index for the SensitiveRegistry; and (iii) `chunks.pkl`, the corresponding RAG knowledge-base chunks. The model card documents the full 9-D feature schema, intervention policy Π, threshold derivation on a 5×5 (λ_s, λ_o) grid, calibration metrics (ECE = 0.013, Brier = 0.041), performance across all seven evaluation settings with bootstrap CIs, and known limitations including the CONFIRM/REVISE coverage gap.

Dataset: nexus-synthetic. Ships as a single JSON array (`full_synthetic.json`, 705 kB, 428 records). Each record includes a natural-language request, a structured multi-step plan with per-tool side-effect and permission metadata, binary safety labels, 4-class intervention labels (`allow`: 82, `block`: 235, `request_confirmation`: 50, `request_revision`: 60), and one of 9 risk-type annotations. Construction: 420 deterministic templates (`seed=0`) plus 8 harvested real plans. Splits are reconstructed deterministically via `src/runtime_safety/benchmarks/split.py`: `seed=42` for train/test (300/128), `seed=7` for train/calibration (240/60).

Dataset: nexus-multistep. Ships as `multistep_sessions_full.json` (616 kB, 120 sessions). Each session contains 2–4 turns, each with its own structured plan identical in schema to `nexus-synthetic`. A `critical_turn_idx` annotation marks the turn at which the cumulative session becomes unsafe (`null` for the 25 legitimate-multi-turn controls). Five session-level risk categories: cross-step exfiltration (35), legitimate multi-turn controls (25), repeated unsafe intent (20), escalating privilege (20), and incremental sensitive access (20). A per-turn-only baseline scores $F_1 \approx 0.5$; session memory achieves 95/95 critical turns caught and 25/25 controls allowed ($F_1 = 1.0$).

Dataset: nexus-ipi. Ships as two dataset configs (`v1`, `v2`), each with a `test` split:

- **v1** (200 records): 100 adversarial + 100 matched controls using a single canonical injection template. $F_1 = 0.995$ [0.98, 1.00].
- **v2** (200 records): 5 injection styles $\times$ 20 pairs: *polite_request*, *urgent_authority*, *social_engineering*, *code_fenced_instruction*, *long_context_payload*. $F_1 = 1.000$.

Each adversarial/control pair shares an identical user request and legitimate first-step tool call; only the `context.history` field differs (clean vs. injected), isolating sensitivity to injection payloads carried through retrieved content or tool output. Deterministic generation with `seed=0`.

Dataset: nexus-stress. Ships as both `nexus_stress.jsonl` and `nexus_stress.json` (668 kB, 200 records). Rule-blind adversarial plans authored without matching to the NEXUS rule-set vocabulary, spanning 20 scenario categories across 11 domains. Gold labels cover all four interventions (BLOCK: 71, CONFIRM: 54, REVISE: 40, ALLOW: 35). Difficulty distribution: medium (64), hard (56), ambiguous (48), easy (25), borderline (7). 91/200 examples are flagged `is_intentionally_difficult_edge_case`; 35/200 are `is_safe_but_suspicious_control`. Semantic deduplication enforces TF-IDF cosine < 0.80 between any retained pair; deterministic build with `seed=0`.

Programmatic access. Once the artifacts are uploaded to a public dataset hub at camera-ready time, users will be able to load all datasets via the standard `datasets` library, replacing `<HUB_USER>` with the released namespace:

```
from datasets import load_dataset
synth  = load_dataset("<HUB_USER>/nexus-synthetic", split="full")
multi  = load_dataset("<HUB_USER>/nexus-multistep", split="test")
ipi_v1 = load_dataset("<HUB_USER>/nexus-ipi", "v1", split="test")
ipi_v2 = load_dataset("<HUB_USER>/nexus-ipi", "v2", split="test")
stress = load_dataset("<HUB_USER>/nexus-stress", split="test")
```

The trained scorer will be loadable directly without cloning the source repository:

```
from huggingface_hub import hf_hub_download
import pickle, numpy as np

path = hf_hub_download("<HUB_USER>/nexus-risk-scorer",
                       "risk_scorer.pkl")
ckpt = pickle.load(open(path, "rb"))
model, scaler = ckpt["model"], ckpt["scaler"]

# Example: 9-D feature vector for a plan
features = np.array([[3, 1, 1, 0, 3, 2, 4.0, 0.08, 0]])
```

```
rho = model.predict_proba(scaler.transform(features))[0, 1]
print(f"rho = {rho:.3f}")  # risk score in [0, 1]
```

The same scripts and artifacts are available in the source-code repository linked in the abstract.

U.2 Software and Hardware Requirements

NEXUS is intentionally lightweight so that runtime safety monitoring imposes no infrastructure cost on the agent loop. Table 20 lists the full software stack and hardware envelope. The full pipeline (Steps 0–7 in Table 22) is pure-CPU, fits in under 2 GB of RAM, requires under 50 MB of disk, and uses only standard scientific-Python dependencies (`scikit-learn`, `NumPy`, `SciPy`) plus `faiss-cpu` for the SensitiveRegistry index and `pydantic` for plan validation. A single CPU core is sufficient for end-to-end reproduction; a 4-core machine completes Steps 0–7 in roughly four minutes. The only external dependency is an OpenAI API key, and only for Step 8 (the optional GPT-4o LLM-judge comparison in Table 7); skipping Step 8 removes all third-party API requirements.

Requirement	Specification
Python	≥ 3.10
scikit-learn	≥ 1.3
NumPy	≥ 1.24
faiss-cpu	$\geq 1.7.4$
pydantic	≥ 2.0
SciPy	≥ 1.11
GPU	Not required (Steps 0–7)
CPU	Single core sufficient; 4-core $\approx$ 4 min
RAM	$<$ 2 GB peak
Disk	$<$ 50 MB total
Step 8 only	OpenAI API key (GPT-4o)

Table 20: Environment specification.

U.3 Model Training and Calibration

The risk scorer is a logistic regression classifier over the 9-dimensional feature vector $\phi(P)$ (Appendix B), with Platt calibration (ECE = 0.013)

Split / Operation	Purpose	Size	Seed
Stratified test+val split	Evaluation	191 (test+val)	42
– Test set	Headline metrics	128	–
– Validation set	Threshold selection	63	–
Training set	Model fitting	300	–
– Base-fit (80%)	LR training	240	7
– Calibration (20%)	Platt scaling	60	7
IPI regeneration	Adversarial pairs	200	0
Bootstrap CIs	Confidence intervals	1,000 resamples	0
Permutation test	4-class significance	5,000 permutations	42
R-Judge 5-fold CV	External retrain	564 records	7

Table 21: Data splits and random seeds.

Threshold selection. Intervention thresholds $(\tau_b, \tau_c) = (0.75, 0.70)$ are selected on the 63-instance validation split and reported exclusively on the 128-instance test split. The scorer-gated intervention policy follows an 8-step deterministic cascade (Algorithm 1, Appendix A).

U.4 Reproduction Pipeline

Table 22 describes the full reproduction pipeline executed by a single command.

Step	Name	Output	Approx. Time
0	Validate environment	Dependency check	$<$1s
1	Load benchmarks	Data integrity check	$<$1s
2	Train risk scorer	`risk_scorer.pkl`	$\sim$2s
3	Evaluate synthetic	Table 3 metrics (F1, IntAcc)	$\sim$5s
4	Evaluate IPI	IPI F1, FPR	$\sim$3s
5	Evaluate sessions	Multi-turn session metrics	$\sim$10s
6	Evaluate NEXUS-Stress	Stress-test F1	$\sim$8s
7	Runtime & statistics	Latency, bootstrap CIs	$\sim$30s
8	LLM-judge (optional)	GPT-4o comparison	$\sim$3 min

Table 22: Reproduction pipeline steps.

```
# Full reproduction (Steps 0-8):
pip install -e .
python run_complete/run_all_experiments.py

# Without LLM-judge (Steps 0-7, no API cost):
python run_complete/run_all_experiments.py --skip-llm-judge
```

U.5 Evaluation Protocol

- **Primary metrics** (Table 3): Binary F1, Intervention Accuracy, per-class precision/recall, all on the held-out 128-instance test split.
- **Bootstrap CIs**: 1,000 stratified resamples (seed=0); 95% percentile intervals.
- **Statistical tests**: Exact McNemar (binary); paired permutation test (4-class, 5,000 permutations, seed=42).
- **R-Judge**: 564 usable records (7 leaked excluded from 571); tool metadata inferred via keyword tables; 5-fold CV retrain (seed=7), majority-vote baseline.
- **NEXUS-Stress**: 200 instances, 20 categories $\times$ 11 domains; TF-IDF ≥ 0.80 duplicate rejection at generation time.
- **IPI**: 200 matched safe/injected pairs per version; deterministic seed=0 generation; v2 ablates 5 injection styles.
- **Sessions**: 120 multi-turn sessions; session-level accuracy (95/95 true positives, 25/25 controls allowed).
- **Latency**: Median over 384 evaluations (3 runs $\times$ 128 instances), single CPU core, no batching.
- **AgentHarm**: 176 benign + 176 harmful behaviors (n=352), target functions mapped via the R-Judge keyword classifier (no GPT-4o baseline used).

U.6 Determinism Guarantees

All experiments (Steps 0–7) are fully deterministic on CPU given the specified seeds. Step 8 (GPT-4o comparison) depends on the OpenAI API and may exhibit minor non-determinism across runs due to model serving variability. Results are bit-reproducible for all non-LLM experiments.

V Ethics Statement

The synthetic benchmark contains no personally-identifiable information. The IPI adversarial set uses fabricated credentials (e.g. the dummy API key `sk_live_4xKj9...`) and synthetic URLs (`attacker.example.com`). The IPI templates we release can in principle be used to probe deployed

agents; we follow the release norm of Yuan et al. (2024) and Andriushchenko et al. (2025) in that surfacing classes of failure benefits defenders more than withholding them benefits attackers. NEXUS is positioned for use in evaluation and as one component of a deployed safety stack; we explicitly do not claim that it provides full assurance against adversaries with access to the agent's LLM weights, the tool implementations, or covert side channels (Table 2).

External artifact terms. AgentHarm (Andriushchenko et al., 2025) is distributed under the MIT License with an additional clause restricting use to research that improves the safety and security of AI systems, together with a request not to use its examples for model training. Our use of AgentHarm (§7.4) is limited to evaluation—mapping target functions to NEXUS plan steps—and is consistent with both conditions. R-Judge (Yuan et al., 2024) is released by its authors without a separate dataset license, alongside the peer-reviewed benchmark paper, for the stated purpose of evaluating LLM agent safety awareness; our use (§7.3) is limited to the same evaluation purpose, plus a supplementary 5-fold cross-validation retraining of the risk scorer reported for research analysis only. We do not redistribute either dataset.

W Online Agent Integration Evaluation

To validate that NEXUS functions correctly as a runtime monitor in an online agent loop, we evaluate it on 30 diverse scenarios simulating a function-calling LLM agent generating structured plans in real time. Scenarios span 8 safe (ALLOW), 6 destructive (BLOCK), 7 borderline (CONFIRM), and 9 scope-fixable (REVISE) requests across file, database, network, and security tool categories.

Results. NEXUS achieves perfect binary detection ($F_1 = 1.000$, precision 1.000, recall 1.000): no unsafe plan is allowed through, and no safe plan is over-blocked. The 4-class intervention accuracy is 0.900, with per-class performance: ALLOW ($F_1 = 1.000$), BLOCK ($F_1 = 0.909$), CONFIRM ($F_1 = 0.824$), REVISE ($F_1 = 0.875$). Median latency is 0.345 ms (P95: 0.970 ms), confirming that NEXUS adds negligible overhead to an agent loop. The 3 misrouted cases involve borderline CONFIRM/REVISE boundaries where the correct intervention is ambiguous even to human annotators.

Class	Prec.	Rec.	F_1	n
ALLOW	1.000	1.000	1.000	8
BLOCK	1.000	0.833	0.909	6
CONFIRM	0.700	1.000	0.824	7
REVISE	1.000	0.778	0.875	9
Overall	**4-class acc = 0.900**			30

Table 23: Online agent integration results (n=30). Binary $F_1 = 1.000$; all unsafe plans are correctly intercepted before execution.

X Guardrail Systems Comparison

Table 24 compares NEXUS with existing runtime safety and guardrail systems across six key capabilities. NEXUS is the only system that provides four-class graded intervention with per-decision provenance at sub-millisecond latency.

Y Inter-Annotator Agreement

To assess label stability on NEXUS-Stress, we conduct an inter-annotator agreement study on 30 stratified instances (8 BLOCK, 8 CONFIRM, 8 REVISE, 6 ALLOW). The second annotator is one of the paper's co-authors, who received the same written labeling protocol described in Appendix P (category definitions and the four-way intervention criteria) and independently labeled each instance without seeing the original gold labels; no external recruitment or additional compensation beyond authorship was involved.

System	Plan IR	4-class	Learned	Arg. Insp.	Session	Latency
NeMo Guardrails	✗	✗	✗	✗	✗	50–200 ms
Guardrails AI	✗	✗	✗	*part.*	✗	10–100 ms
AgentSpec	✓	✗	✗	*part.*	✗	1–5 ms
CLAWGUARD	*part.*	✗	✗	✓	✗	2–10 ms
VeriGuard	✓	✗	✗	✗	*part.*	5–20 ms
ToolSafe	*part.*	✗	✓	*part.*	✓	100–500 ms
R-Judge	✗	✗	✓	*part.*	*part.*	1000–3000 ms
GPT-4o Judge	✓	✓	✓	✓	✗	∼1200 ms
NEXUS (ours)	✓	✓	**calib.**	✓	✓	**0.2–0.4 ms**

Table 24: Comparison with existing runtime safety systems. NEXUS is the only system providing four-class graded intervention, per-decision provenance (every BLOCK/REVISE traces to a named rule), and sub-millisecond CPU latency simultaneously. ✓ = full; *part.* = partial; ✗ = absent.

Results. Cohen's $\kappa = 0.688$ (4-class), indicating *substantial agreement* per the Landis–Koch scale. Binary agreement (safe vs. unsafe) is higher: $\kappa = 0.792$. Per-class agreement rates are: ALLOW 83.3%, BLOCK 75.0%, CONFIRM 75.0%, REVISE 75.0%. The 7 disagreements (23.3%) concentrate at the BLOCK/CONFIRM and REVISE/CONFIRM boundaries, where the correct intervention is genuinely ambiguous. This confirms that the gold labels are stable across annotators and that disagreements reflect intrinsic ambiguity rather than labeling noise.

Metric	Value
n (instances)	30
Observed agreement	0.767
Cohen's κ (4-class)	**0.688** (substantial)
Binary κ (safe/unsafe)	0.792
Disagreements	7/30 (23.3%)

Table 25: Inter-annotator agreement on 30 NEXUS-Stress instances. Substantial 4-class agreement ($\kappa = 0.688$) confirms label stability.

Z AI Usage Statement

Generative AI assistants were used to support drafting and editing of the manuscript and to accelerate routine code-development tasks, such as writing docstrings, adding type hints, and performing minor refactors. All technical claims, experimental designs, empirical results, ablation choices, and figures were defined, executed, verified, and finalized by the authors. No AI assistant was used to fabricate, alter, or synthesize experimental results; every reported value is computed from the released code and benchmarks. The trained risk scorer, rule set, argument inspector, IPI templates, and multi-turn benchmark were authored and validated by the authors.