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

mdelias.hossain@ucf.edu

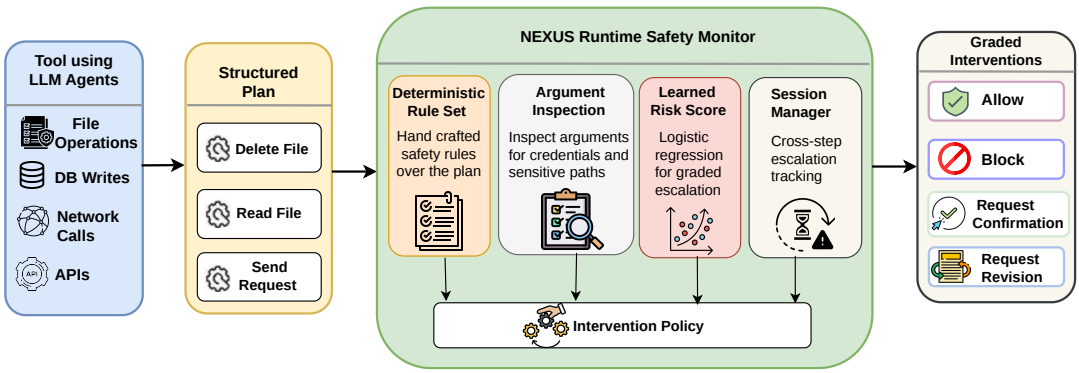

Figure 1: Overview of the NEXUS runtime safety monitor. **(a)** A tool-using LLM agent issues calls across heterogeneous side-effect categories (file operations, database writes, network calls, external APIs) and emits them as a **(b)** *structured plan*, an ordered intermediate representation in which each step carries its tool name, arguments, required permissions, side-effect class, irreversibility flag, sensitivity flag, and estimated cost. **(c)** NEXUS intercepts the plan before any tool is executed and evaluates it through four complementary components: a *deterministic rule set* that fires on nine plan-level safety rules covering permissions, irreversible actions, sensitive access, network egress, budget limits, suspicious patterns, scope ambiguity, broad-scope operations, and untrusted-input-driven actions; an *argument inspector* that scans step arguments for credentials, sensitive paths, sensitive tables and fields, and suspicious URLs; a *calibrated learned risk score* $\rho(P) \in [0, 1]$ produced by a Platt-scaled logistic regression over a nine-dimensional plan feature vector and used as a graded escalation signal; and a *session manager* that tracks exposed entities, cumulative permissions, repeated unsafe attempts, and cumulative cost across turns. **(d)** A formal intervention policy Π combines these signals using critical-violation counts, severity thresholds, and the calibrated thresholds (τ_b, τ_c), and routes each plan to one of four *graded interventions*: ALLOW (execute as proposed), BLOCK (refuse and halt), REQUEST CONFIRMATION (escalate to user approval), or REQUEST REVISION (return the plan for repair). The scorer-gated demotion step, in which $\rho(P)$ arbitrates between BLOCK and CONFIRM when exactly one critical violation fires, is the source of NEXUS's $+27.3$ pp gain in 4-class intervention accuracy over rule-only monitoring.

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,

*Corresponding author: mdelias.hossain@ucf.edu

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 $F_1 = 0.881$, highlighting the difficulty of fine-grained intervention routing. With 0.205 ms median latency, NEXUS adds under 0.1% overhead to typical agent loops. Code, benchmarks, and the calibrated risk scorer are publicly released.

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 can arise from the agent's execution plan itself, including the tools it selects, the inputs it provides, the permissions it requests, and the side effects it produces. For example, an agent may read a sensitive file and then send its contents to an external location. Each step may seem reasonable on its own, but together they form a harmful 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). Thus, agent safety 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. However, these approaches often operate at a single abstraction level, provide limited support for plan-level escalation, or reduce safety to a binary safe/unsafe decision. In practice, some plans should be blocked, some should require confirmation, and others should be revised instead of fully rejected.

We present NEXUS, a structured runtime safety monitor for tool-using language agents. NEXUS operates over a plan representation containing ordered tool invocations, arguments, permissions, side-effect categories, sensitivity indicators, 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 safety rules, argument-level inspection, and calibrated risk scoring. The learned component does not replace the rule layer; it provides a calibrated escalation signal for fine-grained intervention 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. The evaluation shows that plan-level monitoring is effective when unsafe behavior is visible in tool actions, arguments, permissions, or side effects. It also identifies an important boundary: when harmful and benign behaviors share the same structural tool plan and differ mainly in latent intent, additional prompt-aware or intent-aware safeguards may be required (Zhong et al., 2025). Our main contributions are as follows:

- **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), AgentDrift (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: our IPI evaluation (Appendix H) shows that once an injection steers the agent, the resulting plan is caught by any structured-plan monitor.

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

The framework operates over a *structured plan IR* and combines three logically independent safety signals (rules, argument inspection, calibrated risk) under a single intervention policy Π. Each signal is testable in isolation, and any decision can be traced back to a named rule, an argument-inspector finding, or a calibrated threshold crossing.

Figure 1 summarizes the NEXUS runtime pipeline. A tool-using LLM agent first produces a structured plan rather than directly executing tool calls. This plan contains the selected tools, their arguments, required permissions, side-effect categories, and execution context. NEXUS intercepts the plan before execution and evaluates it through four complementary components: a deterministic rule set for explicit safety violations, an argument inspector for sensitive values and risky inputs, a calibrated learned risk scorer for graded escalation, and a session manager for cross-turn state. The resulting signals are passed to an intervention policy that decides whether to ALLOW, BLOCK, request user CONFIRMATION, or request plan REVISION.

The remainder of this section defines each component. We first introduce the structured plan intermediate representation, then describe the rule set and argument inspector. Section 4.3 presents the calibrated learned risk scorer, and Section 4.4 defines the intervention policy that combines all signals into a single runtime decision.

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

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
AgentDrift (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.

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).

available permissions, remaining budget, prior tool-output history, and per-session policy flags.

4.2 Rules and Argument Inspection

NEXUS applies nine deterministic rules. The first six cover permission checks, irreversible actions, sensitive access, network access, budget limits, and suspicious patterns. During development, we added three additional rules: R7, *scope ambiguous action*, which escalates security-posture changes made through insecure flags (severity HIGH, or CRIT for unbounded grants); R8, *broad scope operation*, which flags wildcard or glob patterns in file and data operations (severity MED); and R9, *external source driven*, which flags cases where untrusted input drives privileged operations (severity HIGH).

Each rule emits violations with severity $\text{sev}(v) \in \{\text{LOW}, \text{MED}, \text{HIGH}, \text{CRIT}\}$. In parallel, the argument inspector scans step arguments for 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 $\phi(P)$ covering plan length, indicators of irreversible / sensitive / delete-side-effect steps, per-category step counts, total estimated cost, and tool-category diversity (full definitions in Appendix B). We use a stratified $70/15/15$ split (seeds 42 / 7) of the 428-instance synthetic benchmark for train, validation, and test; thresholds (τ_b, τ_c) are selected on the validation split and all reported metrics come from the held-out test split exclusively. The validation-best thresholds are $\tau_b = 0.30$, $\tau_c = 0.10$; the paper defaults $(0.75, 0.70)$ yield identical F_1 with slightly higher intervention accuracy on the test split (0.667 vs. 0.651), confirming no threshold leakage. The training set is further split $80/20$ (seed 7) into base-fit and calibration sets, on which we fit Platt scaling and isotonic regression. Platt scaling reduces ECE $\approx 6.5\times$ over the raw scorer (full metrics in Table 9, Appendix C); we use Platt-scaled ρ throughout.

Role in Π. We use ρ as a graded escalation signal rather than as 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 above ensures that (τ_b, τ_c) operate on a probability scale that meaningfully reflects empirical unsafe rates.

Why Platt over isotonic?

Both reduce ECE substantially over the raw scorer, and isotonic obtains a marginally lower Brier score (0.038 vs. 0.041). We deploy Platt because it achieves the *lowest* ECE (0.013, $\approx 6.5\times$ reduction over raw) and is more stable on the small 60-instance calibration split, where isotonic risks overfitting (Appendix C).

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

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

Cases are evaluated top-to-bottom and the first matching action is returned. Multiple critical violations trigger an unconditional block. When exactly one critical violation is present, the risk score gates between blocking the plan ($\rho \geq \tau_b$) and requesting confirmation ($\rho < \tau_b$); this is the scorer-gated demotion step. High-severity violations route to confirmation, medium-severity violations route to revision, and the learned 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.

Why $(\tau_b, \tau_c) = (0.75, 0.70)$?

Selected on the 63-instance validation split and held constant across all 25 (λ_s, λ_o) grid settings (Appendix E). The validation-best $(0.30, 0.10)$ yields identical F_1 but lower test-split intervention accuracy (0.651 vs. 0.667), confirming no threshold leakage from test data into deployment.

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 0.205 ms per plan on the IPI set (4,349 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 generated from 9 deterministic templates over 7 tool categories. A stratified 70/15/15 split with seed 42 yields a 300-instance training set, a 63-instance validation set for threshold selection, and a 128-instance held-out test set 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 contains a natural-language request, an ordered structured plan, a binary safety label, and an intervention label from {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 with the same topic, user, and tool category but without the unsafe step. This split tests whether NEXUS correctly classifies the agent's resulting plan when the surrounding *context* contains adversarial text.

The two synthetic splits evaluate different dimensions: structural detection (synthetic) and ro-

bustness to injection context (IPI). External benchmark generalization is evaluated on R-Judge (Yuan et al., 2024) (Section 7.3) and AgentHarm (Andriushchenko et al., 2025) (Section 7.4); evaluation on organic agent traces is left for future work (Section 9).

6 Main Results

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

On the synthetic split, NEXUS achieves $F_1 = 0.949$ and 4-class intervention accuracy of 0.641. This matches rule-only detection in binary F_1 but substantially improves intervention routing over rule-only monitoring (0.367 to 0.641). On R-Judge, NEXUS reaches $F_1 = 0.861$ $[0.83, 0.89]$, compared with 0.849 for rule-only. The gain is concentrated in additional unsafe detections, especially in IoT trajectories after enabling the permanence-flag rule in §7.3. Per-category performance on R-Judge is strongest for Finance, Program, and Web ($F_1 \geq 0.89$), followed by Application (0.852), with IoT remaining the hardest category (0.518).

On AgentHarm (n=352), both NEXUS and rule-only obtain $F_1 = 0.591$ $[0.53, 0.65]$. This occurs because paired harmful and benign examples often share the same target tools, so the structural plan does not contain the unsafe intent signal. We therefore treat AgentHarm as a threat-model boundary rather than as a regression.

Statistical significance. On the synthetic split, binary McNemar testing gives $p = 1.0$ because rule-only and NEXUS make the same binary decisions. The scorer's effect appears in the 4-class intervention task. A paired permutation test over intervention accuracy (5,000 permutations) gives $\Delta = +0.273$, $p < 0.001$: the scorer moves $35/128$ instances from CONFIRM to BLOCK, and all 35 have gold label BLOCK. On R-Judge, binary McNemar gives $\chi^2 = 3.125$, $p = 0.077$, corresponding to 7 additional unsafe detections. Thus, NEXUS differs most clearly from rule-only in intervention accuracy, per-category behavior, and R-Judge generalization, not in IPI or AgentHarm.

	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.

Scorer-gated demotion adds +27.3 pp

Scorer-gated demotion adds +27.3 pp 4-class intervention accuracy over rule-only monitoring ($0.367 \rightarrow 0.641$, permutation $p < 0.001$, 5,000 permutations), while preserving identical binary $F_1 = 0.949$ and 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 shows its largest gains when unsafe behavior appears inside tool arguments or plan structure. Compared with rule-only monitoring, NEXUS improves *sensitive data access* from $F_1 = 0.273$ to 1.000, because argument inspection detects credentials and sensitive paths that category-level rules miss. Compared with learned-only monitoring, NEXUS improves *budget abuse*, *ambiguous confirm*, and *ambiguous revise* by $F_1 = +1.000$ each. This shows that a scalar risk score alone does not reliably capture structural signals such as exceeded budgets or under-specified 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. To test whether session context resolves this failure mode, we evaluate NEXUS on a session-aware benchmark of 120 ordered sessions: cross-step exfiltration (35), esca-

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: $[0.864, 1.000]$.

lating 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: an exposed-entities set, a cumulative permission set, a repeated-unsafe-attempt counter, and a cumulative-cost accumulator. No additional learned component is added. Table 4 shows that session memory raises critical-turn detection from $0/95$ to $95/95$ across the unsafe sessions, with zero false positives on the 25 legitimate controls. This result suggests that the multi-step failure is mainly due to missing session state, not to the rule vocabulary alone.

Multi-turn gap: $0/95 \rightarrow 95/95$

Session memory closes the multi-turn detection gap entirely: $0/95 \rightarrow 95/95$ critical turns caught, 0 false positives on 25 legitimate controls. No additional learned component was required; lifting four lightweight session-state features into Π's CONFIRM branch is sufficient.

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

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.

Results. Table 5 shows that NEXUS reaches $F_1 = 0.861$, improving over rule-only by $+0.012$ while remaining below the synthetic split result (0.949), as expected under distribution shift. The gain is mainly 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 versus 0.848 for the synthetic-trained baseline, indicating that the feature representation captures transferable safety signal.

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 improves 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. Thus, R-Judge supports external validity for structured tool-call monitoring while highlighting the need for richer domain-specific 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 example includes a free-text prompt and a list of target functions. We map each target function to a NEXUS `PlanStep` using the same keyword-based classifier used for R-Judge, but we do not include the prompt text in the plan.

On this 352-instance test set, NEXUS obtains $F_1 = 0.591$ $[0.53, 0.65]$, with precision 0.508 and recall 0.705. Rule-only monitoring obtains the same F_1. This result is expected because Agen-

tHarm often pairs harmful and benign examples that use the same target tools. In such cases, the unsafe signal is mainly expressed in the prompt intent, not in the structural tool sequence.

Scope analysis. AgentHarm therefore tests a boundary of plan-level monitoring. NEXUS is designed to detect unsafe structure in plans, arguments, permissions, and side effects; it is not designed to infer latent harmful intent from the prompt alone. When harmfulness depends on prompt-level intent while the tool plan remains structurally similar, prompt-aware or argument-aware methods such as ToolSafe (Mou et al., 2026) are the more appropriate comparison class. The identical performance of NEXUS and rule-only monitoring further supports our design claim: the learned risk score is used for borderline intervention routing, not for latent-intent detection.

→ AgentHarm parity: a structural boundary

When paired harmful and benign behaviours share the same structural tool plan, the unsafe signal lives in prompt-level intent, not in the plan; plan-level monitors of any kind are structurally blind here. NEXUS is positioned as one layer of a defence stack, complementary to prompt-aware safeguards.

7.5 Runtime Efficiency

Table 6 reports the runtime cost of NEXUS on the 128-instance synthetic test split. We repeat the evaluation three times, yielding 384 total decisions on a single CPU core without batching. NEXUS is lightweight: it achieves a median latency of 0.205 ms and a P99 latency of 0.456 ms, corresponding to approximately 4,349 decisions per second. This overhead is negligible compared with a typical LLM agent turn, which often takes hundreds or thousands of milliseconds. Therefore, NEXUS can be inserted before tool execution without requiring a GPU or noticeably slowing the agent loop.

0.205 ms median latency, no GPU

Under 0.1% overhead on a typical agent loop; dominant cost is `predict_proba` on a 9-D logistic regression.

7.6 Comparison with LLM Judges

We compare NEXUS with GPT-4o on the same 200-instance NEXUS-Stress benchmark, treating GPT-4o as a strong LLM-judge baseline. As shown in Table 7, GPT-4o obtains higher binary detection and intervention accuracy, but at substantially higher latency. NEXUS reaches 92% of GPT-4o's binary F_1 and 78% of its intervention accuracy while running approximately 4,800× faster. GPT-4o also struggles with fine-grained intervention routing, with per-class F_1 of 0.455 for CONFIRM and 0.405 for REVISE. This suggests that the hardest part of NEXUS-Stress is not only unsafe-plan detection, but deciding the correct intervention level.

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.

Monitor	Binary F_1	Int.-Acc	Latency
GPT-4o	0.956	0.555	∼1,200 ms
NEXUS	**0.881**	**0.435**	**0.205 ms**
NEXUS / GPT-4o	92%	78%	∼4,800× **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 0.25 ms NEXUS latency; the measured median latency is 0.205 ms.

7.7 Adversarial Stress Testing

We evaluate NEXUS on **NEXUS-Stress**, a 200-example adversarial benchmark designed without matching the rule-set vocabulary. The benchmark covers 20 scenario categories across 11 domains; Appendix P provides the construction and audit details.

NEXUS achieves binary $F_1 = 0.881$, with precision 0.829 and recall 0.939. Table 8 provides the 4-class confusion matrix, which gives a more detailed view of intervention behavior. Among the 94 gold CONFIRM and REVISE examples, NEXUS frequently predicts CONFIRM (96 predictions vs. 54 gold). This shows a conservative pattern: when

gold ↓ / pred →	ALLOW	BLOCK	CONFIRM	REVISE
ALLOW ($n = 35$)	**19**	3	13	0
BLOCK ($n = 71$)	6	**30**	35	0
CONFIRM ($n = 54$)	2	4	**48**	0
REVISE ($n = 40$)	15	0	25	**0**

Table 8: NEXUS-Stress 4-class confusion matrix. NEXUS shows conservative behavior on ambiguous cases by often selecting CONFIRM, but it does not predict REVISE, indicating a middle-severity coverage gap.

the system is uncertain, it tends to ask for user confirmation instead of blocking automatically.

The main limitation is that NEXUS does not predict REVISE on rule-blind inputs. This exposes a middle-severity coverage gap, where the plan is not safe enough to allow but does not clearly match existing revision-triggering rules. We discuss this limitation in §9, with per-category details in Appendix P.

7.8 Additional Results in the Appendix

The appendix reports supporting analyses that complement the main results: 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 analysis (Appendix J), error analysis (Appendix K), and the full NEXUS-Stress breakdown (Appendix P). The key findings from these analyses 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. Once agents invoke tools, safety-relevant signals appear in tool choices, arguments, permissions, side effects, and execution order. NEXUS uses this structured plan representation to inspect executable behavior before tool calls are committed and to ground interventions in explicit rules, argument findings, or calibrated risk scores.

The results show that the hybrid design is useful. Rules enforce hard constraints, argument inspection detects sensitive values inside tool inputs, and the calibrated scorer improves intervention routing. In particular, the scorer mainly helps distinguish BLOCK from CONFIRM, rather than serving as the primary unsafe-plan detector.

The main challenge is fine-grained intervention selection. Binary unsafe detection is easier than deciding whether to BLOCK, CONFIRM, or REVISE. The NEXUS-Stress results show that medium-severity and rule-blind cases remain difficult, especially when revision is more appropriate than confirmation.

The R-Judge and AgentHarm results clarify the boundary of plan-level monitoring. 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 mainly on latent prompt intent. Thus, NEXUS should complement prompt-aware and intent-aware safeguards rather than replace them.

Finally, the sub-millisecond CPU latency makes NEXUS practical as a runtime governance layer before tool execution. Its modular design allows rules, argument checkers, and thresholds to be updated without retraining the full system.

9 Conclusion

We presented NEXUS, a runtime safety monitor for tool-using LLM agents that combines nine 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: while rules provide perfect recall, the scorer improves 4-class intervention accuracy by $+27.3$ pp by distinguishing BLOCK from CONFIRM when a critical rule fires. 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, with the permanence-flag rule raising IoT performance from 0.462 to 0.518. On AgentHarm, NEXUS matches rule-only performance ($F_1 = 0.591$), highlighting a deliberate threat-model boundary when harmful and benign behaviours share target tools. On IPI, both monitors achieve 0% ASR at 99% control allow, showing robustness to injection text at the structured-plan level. On NEXUS-Stress, NEXUS reaches $F_1 = 0.881$ and remains competitive with GPT-4o while requiring only 0.205 ms median latency and zero marginal cost. Future work should expand medium-severity rule coverage to improve CONFIRM/REVISE routing and validate session-aware multi-turn monitoring on organic agent traces. We release the code, benchmarks, retrained 9-D risk

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 ($F_1 = 0.881$) provide stronger out-of-distribution evidence, but live-trace evaluation is still needed. NEXUS currently supports seven tools; 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 are difficult: Π predicts no REVISE on 40 gold REVISE examples and defaults to CONFIRM. Future rules should target scope-tightening and disambiguation patterns.

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. Agentdrift: Unsafe recommendation drift under tool corruption hidden by ranking metrics 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 synthetic test split.
- **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 W** 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. 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.

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 expected calibration error (ECE) compared with the raw logistic-regression probabilities. We use Platt scaling as the default because it provides strong calibration while being more stable on the small calibration split. Although isotonic regression obtains a slightly lower Brier score, Platt scaling offers the best ECE among the reported settings and avoids overfitting risk on limited calibration data.

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)
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 $\exists\, v \in V : \text{sev}(v) = \text{HIGH}$ **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 $(\text{ALLOW}, \text{JUSTIFY}(\emptyset, \rho))$

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 achieves the lowest ECE among the reported settings and is less prone to overfitting on the small calibration split, while isotonic regression obtains the lowest Brier score.

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

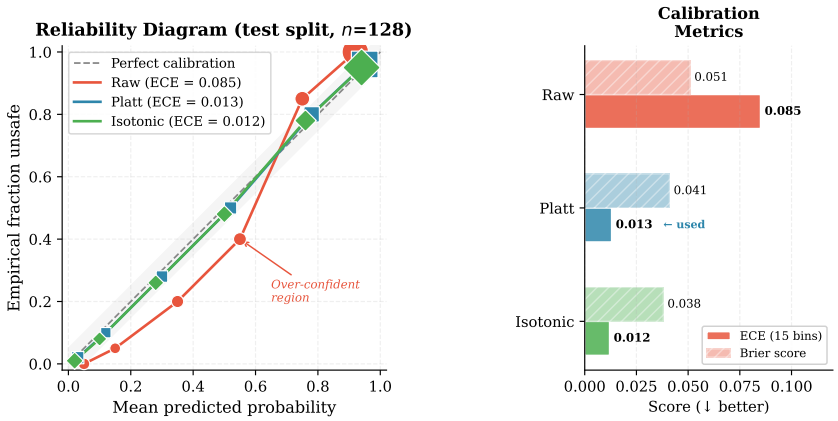

Figure 2: Reliability diagram for the learned risk scorer on the 128-instance held-out test split. The diagonal line indicates perfect calibration, and marker size reflects the number of examples in each bin. The raw scorer is over-confident in the mid-probability range, while post-hoc calibration moves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

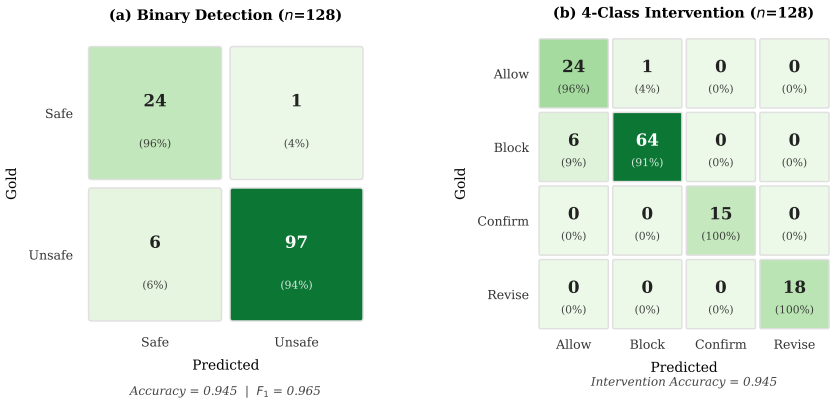

Figure 3: Confusion matrices for NEXUS on the 128-instance synthetic test split. The binary matrix shows safe/unsafe detection performance, while the 4-class matrix shows intervention routing across ALLOW, BLOCK, CONFIRM, and REVISE. Errors are concentrated in gold BLOCK cases predicted as ALLOW.

and Brier score in Table 9 as the primary calibration summaries.

D Confusion Matrices

Figure 3 reports the binary and 4-class confusion matrices for NEXUS on the 128-instance synthetic test split. In the binary setting, NEXUS correctly identifies 97 of 103 unsafe plans and allows 24 of 25 safe plans. The remaining errors consist of one safe plan flagged as unsafe and six unsafe plans predicted as safe. In the 4-class setting, NEXUS correctly routes all CONFIRM and REVISE cases, while most residual error is concentrated in the BLOCK class, where six gold BLOCK plans are predicted as ALLOW. This pattern shows that the main failure mode is not confusion between confirmation and revision, but missed high-severity cases with weak structural signals.

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 peaks at 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. 0.944) with slightly lower intervention accuracy (0.651), confirming no threshold leakage

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 $c(\text{ALLOW}){=}0$, $c(\text{REVISE}){=}0.1$, $c(\text{CONFIRM}){=}0.3$, $c(\text{BLOCK}){=}1$. 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.

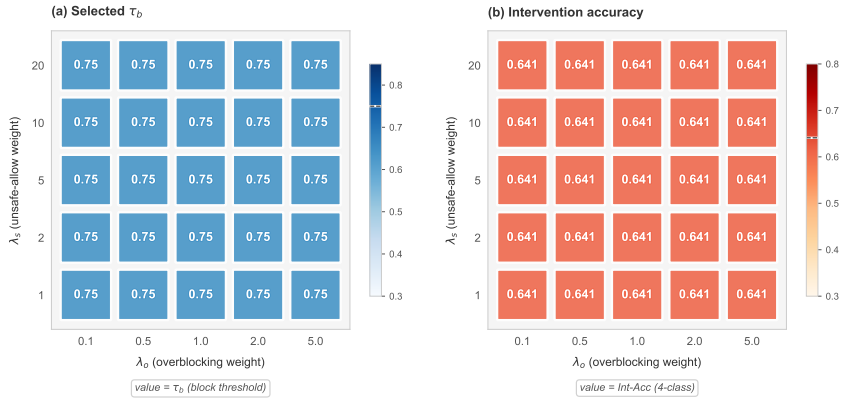

Figure 4: Loss-optimal (τ_b, τ_c) across the 5×5 (λ_s, λ_o) grid. **(a)** The selected τ_b is constant at 0.75 across all 25 grid cells. **(b)** 4-class intervention accuracy is correspondingly constant at 0.641 at the selected operating point.

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.

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 $\tau_b{=}0.70$; 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

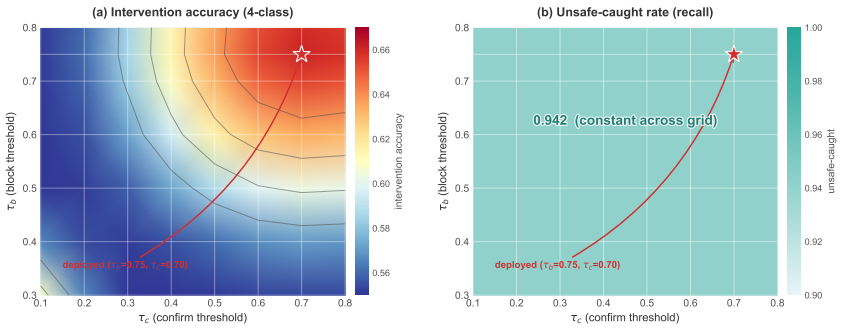

Figure 5: Threshold sweep over the (τ_b, τ_c) grid. (a) Intervention accuracy peaks at 0.641 at the deployed $(0.75, 0.70)$ operating point (red star). (b) The unsafe-caught rate is constant at 0.942 across the entire grid, because rule and argument-inspection violations fire on every unsafe plan regardless of the learned-score threshold.

Operating point	τ_b	τ_c	F_1	Int.-Acc (test split)
paper default	0.75	0.70	0.949	**0.667**
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 higher test-split intervention accuracy than the validation-best $(0.30, 0.10)$, confirming no threshold leakage.

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 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)

Model	F_1	Prec.	Rec.	Int.-Acc
LogReg, 9d	0.949	0.990	0.942	–
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.

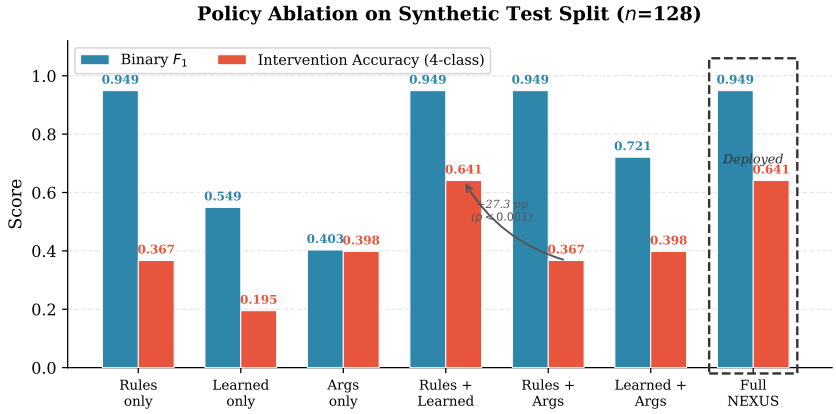

Figure 6: Policy ablation on the 128-instance synthetic test split. Binary F_1 measures unsafe-plan detection, while 4-class intervention accuracy measures routing across ALLOW, BLOCK, CONFIRM, and REVISE. Rules provide most of the binary detection performance, whereas the learned risk scorer improves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

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 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.

Configuration	F_1 [95% CI]	Unsafe-caught	Overblock	Int.-Acc
Rules only	0.949 [0.92, 0.98]	0.942	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]	0.942	0.040	0.641
Rules + args	0.949 [0.92, 0.98]	0.942	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]	0.942	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 $p < 0.001$).

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 $0/100 \in [0.000, 0.037]$.

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.

(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

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.

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.

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) Six *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 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

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.

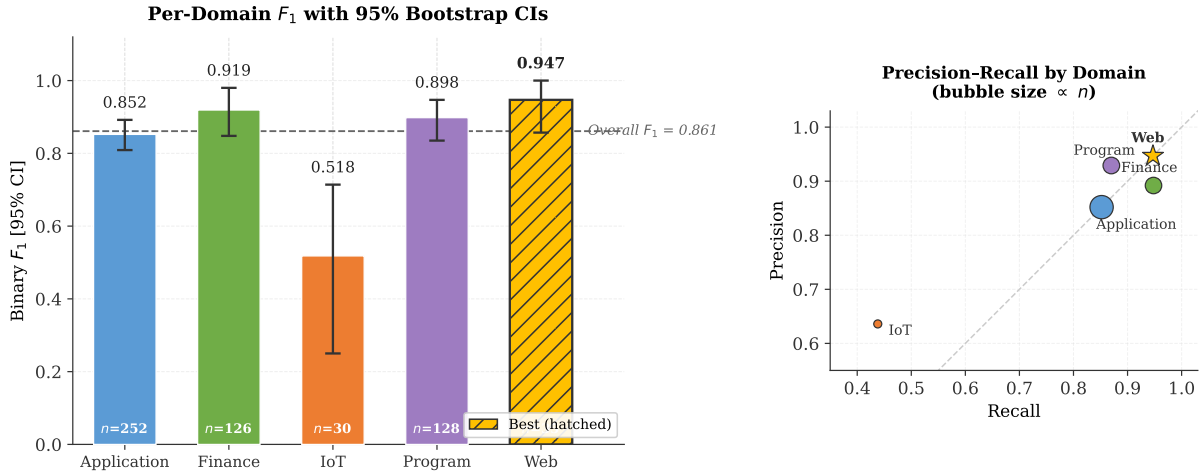

Figure 7: Per-category R-Judge results for NEXUS with 95% bootstrap CIs (1,000 resamples). `rule_only` differs by at most 0.007 F_1 per category and is not shown. NEXUS performs well on tool-heavy categories (Finance, Program, Web) where the unsafe signal is in a parseable transmit/delete/transfer action;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

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.

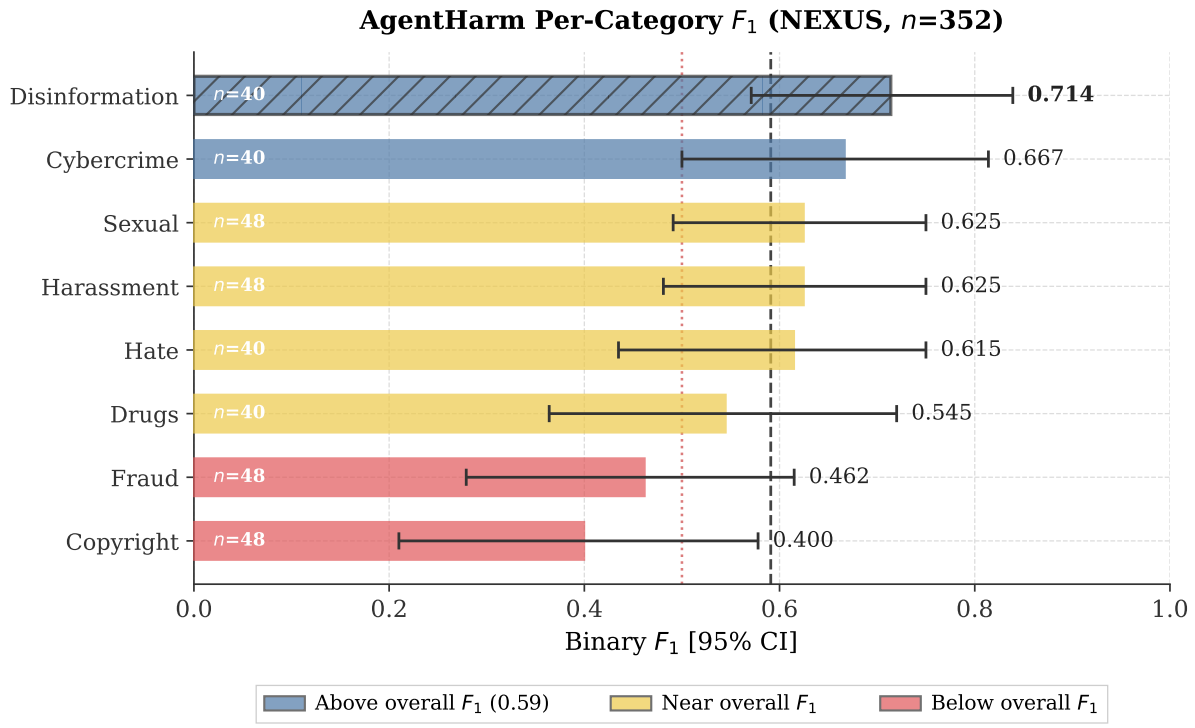

Figure 8: AgentHarm test-public split (paired harmful + benign, n=352). Binary F_1 with 95% bootstrap CIs (1,000 resamples, seed 0). NEXUS and `rule_only` score identically because the structural plan is shared between paired harmful and benign behaviours; the per-category spread reflects how often the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

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.

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

metric	point	95% bootstrap CI
binary F_1	0.881	[0.790, 0.881]
binary precision	0.829	[0.841, 0.938]
binary recall	0.939	[0.728, 0.849]
binary accuracy	0.745	[0.685, 0.800]
4-class interv-acc	0.435	[0.350, 0.490]
macro F_1 (4-class)	0.257	[0.217, 0.294]
unsafe-allow rate	0.212	N/A
overblock rate	0.457	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).

category	n	bin-F_1	int-acc	overblock
destructive actions	14	1.000	0.786	0.000
sensitive data access	14	1.000	0.429	0.000
privilege misuse	10	1.000	0.500	0.000
external comm/exfiltration	13	0.960	0.692	0.000
indirect prompt injection	10	1.000	0.700	0.000
chained benign→unsafe	8	1.000	0.625	0.000
multi-turn escalation	10	0.947	0.600	1.000
low-signal unsafe	8	0.933	0.375	0.000
ambiguous intent (CONFIRM)	16	0.769	0.000	0.000
medium-risk reversible (REVISE)	15	0.571	0.000	0.000
budget/resource abuse	8	0.545	0.250	0.000
IoT permission/time-window	8	0.545	0.125	0.000
benign suspicious-looking safe	20	0.000	0.350	0.650
safe aggregation/minimization	12	0.000	0.833	0.167

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. The categories with low 4-class accuracy (ambiguous intent, medium-risk reversible, IoT) are those whose gold labels require CONFIRM or REVISE, which NEXUS conservatively collapses to CONFIRM or ALLOW on rule-blind inputs.

`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). The benchmark surfaces a structural diagnostic: NEXUS over-assigns CONFIRM (96 predicted vs. 54 gold) and never predicts REVISE, the conservative fallback of scorer-gated demotion on rule-blind inputs.

→ Rule-blind inputs: Π collapses to CONFIRM

This is a structural diagnostic, not a bug. When no rule fires, the policy has no medium-severity signal to trigger REVISE; the calibrated scorer can only push borderline cases to CONFIRM. The fix is additional middle-severity rules (or a learned routing head), not more aggressive thresholds.

Per-category breakdown. Table 18 reports binary F_1 and 4-class intervention accuracy across the 20 scenario categories. NEXUS performs at $F_1 \geq 0.93$ on the categories where the rule set is most aligned (destructive actions, privilege misuse, indirect prompt injection, low-signal unsafe, multi-turn escalation) and underperforms on categories that require CONFIRM or REVISE intervention classes (ambiguous intent, medium-risk reversible, IoT permission/time-window, budget/resource abuse).

Per-difficulty. NEXUS achieves $F_1 = 0.91$ on *hard* (n=56) and $F_1 = 0.84$ on *ambiguous* (n=48), with 4-class intervention accuracy of 0.82 and 0.00 respectively; the gap reflects the CONFIRM over-

prediction effect on ambiguous cases. On the 7 *borderline* cases $F_1 = 0.92$, intervention accuracy 0.00.

Top failure modes. The 4-class errors fall into three main patterns. First, NEXUS sometimes over-intervenes when a high-severity rule fires on a plan whose gold label requires only CONFIRM. Second, some unsafe plans are allowed because the relevant risk appears in rule-blind argument content rather than in explicit per-step structure. Third, the rule set occasionally treats scope-tightening opportunities as critical violations, producing stronger interventions than necessary. These errors suggest that future versions should add more precise middle-severity rules, especially for distinguishing CONFIRM from REVISE and for handling ambiguous but recoverable plans.

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 public 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 Dataset License and Usage Terms

The NEXUS benchmarks consist of synthesized plan templates spanning multiple tool categories (file operations, database writes, network calls, and external APIs). All datasets are released under **CC BY 4.0** and the trained risk scorer under **Apache-2.0**. By downloading or using any of these artifacts you agree to the following terms.

- **Research use only.** The datasets are released *solely for research on agent safety* (monitoring, evaluation, red-teaming, and policy design). Use for offensive security against systems without explicit authorization, or to train models intended to circumvent safety monitors, is out of scope and not authorized.
- **Dataset cards are authoritative.** Each Hugging Face dataset card documents per-dataset construction details, known limitations, sensitive-content disclosures, and any additional usage constraints. Where this paper and the dataset card disagree, the dataset card governs.
- **Synthetic and curated content.** Sensitive-looking arguments inside the datasets (credentials, paths, tables, URLs) are synthetic test fixtures. They are designed to exercise the argument inspector and rule set; they do not refer to real systems, users, or accounts.
- **Attribution.** If you use the datasets, model, or rule set in published work, please cite this paper and link to the relevant Hugging Face dataset card.
- **Redistribution.** You may redistribute the datasets and derived artifacts under the same CC BY 4.0 terms (or, for the scorer, Apache-2.0), preserving attribution and this license notice.

The dataset cards live at the URLs listed in §U.2; consult each card before downloading.

U.2 Public Release on Hugging Face

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.

Table 19: Hugging Face release inventory.

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

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 Hugging Face 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. All datasets are loadable via the Hugging Face `datasets` library:

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

The trained scorer can be loaded directly without cloning the source repository:

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

```
path = hf_hub_download("EliasHossain/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]
```

U.3 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.

Table 20: Environment specification.

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)

U.4 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)

Table 21: Data splits and random seeds.

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

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 a 7-step deterministic cascade (Algorithm 1, Appendix A).

U.5 Reproduction Pipeline

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

Table 22: Reproduction pipeline steps.

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`	∼2s
3	Evaluate synthetic	Table 3 metrics (F1, IntAcc)	∼5s
4	Evaluate IPI	IPI F1, FPR	∼3s
5	Evaluate sessions	Multi-turn session metrics	∼10s
6	Evaluate NEXUS-Stress	Stress-test F1	∼8s
7	Runtime & statistics	Latency, bootstrap CIs	∼30s
8	LLM-judge (optional)	GPT-4o comparison	∼3 min

```
# 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.6 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**: 44 benign + 66 harmful behaviors; evaluated against GPT-4o safety baseline.

U.7 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).

W 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.