

Edge-Native Semantic Firewall for Autonomous LLM Agents: A Structured Chain-of-Thought Verification Framework

Sushant Poudel

*Dept. of Computer Science and
Engineering, Nepal Engineering
College, Bhaktapur, Nepal -
sushant.poudel2028@gmail.com*

Rakhee Pandey

*Dept. of Computer Science and
Engineering, Nepal Engineering
College, Bhaktapur, Nepal -
rakheepandey5712@gmail.com*

Aashika Pandey

*Dept. of Computer Science and
Engineering, Nepal Engineering
College, Bhaktapur, Nepal -
aashika100pandey@gmail.com*

Abstract- An autonomous agent that executes actions rather than proposing them sits outside the reach of role-based access control, which authenticates an identity but not intent. Routing every proposal to a cloud-hosted frontier model closes that gap but adds round-trip latency and, in regulated settings, is often prohibited outright. We describe an edge-native semantic firewall: a 3.8B-parameter Phi-3-mini model, 4-bit quantized under a 4.2 GiB VRAM ceiling, acting as the evaluator in a Generator-Evaluator pipeline on one consumer laptop. Its mechanism is a structured Chain-of-Thought JSON schema that requires the evaluator to name the governing rule and justify the match before emitting a decision, turning an opaque verdict into an auditable trace. We evaluate it on a 600-scenario corpus across three policy rules, including 155 adversarial scenarios spanning eight prompt-injection techniques, 60 compound and 30 boundary cases; all 1,800 generations ran locally at temperature 0. The results invert a natural assumption: constraining output format without requiring the reasoning step produced the least safe evaluator of the three. The JSON-only arm approved 46.2% of proposals the policy would block or route to human review, worse than unconstrained free-form at 17.2%. The full schema cut that to 23.5% and raised decision accuracy from 52.3% to 66.3%. On the irreversible class, accuracy runs 62.5% under JSON-only, 72.1% under free-form, and 90.8% under the proposed schema, with hard-denial approvals falling from 71 to 6. The gain is real and not sufficient: the best configuration still approved 6 of 208 hard-denial actions, and was more permissive than free-form on ambiguous proposals that should have reached a human. Peak VRAM was 3.95 GiB and median latency 2.55 s. An edge model of this size can serve as one layer of a defence-in-depth

stack; the evidence does not support treating it as a sole control.

Keywords- *Agentic AI Security; Edge Inference; Semantic Verification; Chain-of Thought Prompting; Data Sovereignty*

I. INTRODUCTION

Agentic LLM systems can execute actions through tools, including database queries, code modifications, and infrastructure operations [1]–[6]. Unlike conventional assistants, incorrect actions can have irreversible consequences, motivating verification before execution

RBAC verifies identity but does not determine whether a specific proposed action is safe or policy-compliant [7]. We therefore introduce an LLM agent firewall that evaluates actions against explicit policies before execution.

A cloud-hosted evaluator can catch subtle violations, but repeated round trips add latency and may conflict with data-sovereignty requirements in regulated environments.

This paper evaluates a local firewall based on a structured Chain-of-Thought JSON schema that requires the model to identify the governing rule, justify the match, assign confidence, and emit the decision last. The output grammar enforces this ordering

A. Contributions

- A localized Generator–Evaluator pipeline with sequential memory management that keeps both agent roles inside a 4.2GiB VRAM budget on consumer hardware, measured at 3.95GiB peak during evaluation.
- A structured Chain-of-Thought JSON schema that constrains the field ordering of the evaluator’s output, with the decision emitted last.

- A reproducible 600-scenario evaluation corpus in which ground-truth labels follow from the policy by construction rather than from a scorer sharing vocabulary with the evaluator, stratified across three rules and tagged adversarial, compound, and boundary.
- A full execution of that corpus on the real model under three prompting conditions (1,800 generations), reporting per-rule accuracy, adversarial-subset performance by technique, a quantified baseline without the reasoning scaffold, and latency and throughput on edge hardware.
- An ablation showing that structured output without reasoning is the least safe condition, with 46.2% unsafe approvals versus 17.2% for free-form text.

II. RELATED WORK

A. Autonomous Agentic Workflows

Brown et al. [1] established that large language models generalise from few examples, and Ouyang et al. [2] showed that this capacity can be shaped toward preferred behaviour through feedback-driven fine-tuning.

Toolformer [3] and ReAct [4] extended the pattern by letting models invoke external tools and interleave reasoning with environmental action. AutoGen [5] formalised multi-agent orchestration, and Park et al. [6] showed that persistent memory and social coordination among LLM agents produce emergent, goal-directed behaviour. Cemri et al. [8] then built a taxonomy from over a thousand annotated multi-agent traces and found that a large share of real-world failures trace back to unverified inter-agent action. That finding is the direct motivation for inserting a verification stage rather than trusting orchestration alone.

B. Guardrail Frameworks and Content Moderation

A substantial body of work places a classifier between the model and the world. Llama Guard [9] and ShieldGemma [10] label prompts and responses against a fixed harm taxonomy. NeMo Guardrails [11] is closer to the present work: it supports programmable rails expressed in a domain-specific language, so application authors can constrain dialogue flow and define custom checks. The distinction we draw is that these

systems evaluate content against a taxonomy, whereas the firewall evaluates a proposed action against an operator-authored policy that maps action classes to routing decisions. A harm classifier can tell you a sentence is abusive. It cannot tell you whether `systemctl restart nginx` is permitted by your change-management policy at this hour. The two are complementary and would ordinarily be deployed together.

C. LLM-as-a-Judge and Real-Time Verification

Zheng et al. [12] formalised using one language model to evaluate the outputs of another. Moving that paradigm into a real-time security path exposes a tension the original framing does not have: round-trip latency compounds across sequential operations, and transmitting the actions themselves to an external API can be the data-sovereignty violation the deployment was meant to avoid. Brundage et al. [13] argue that verifiable, auditable claims should be a design requirement for consequential AI deployments rather than opaque trust in a vendor's evaluator. The auditable CoT trace in this work is our attempt to satisfy that requirement without leaving the host.

D. Agent Security and Prompt Injection

Prompt injection against tool-using agents is a measured phenomenon, not a hypothetical. Greshake et al. [7] showed that indirect injection through retrieved content is practical. DeBenedetti et al. [14] built AgentDojo to evaluate injection attacks and defences dynamically; Zhan et al. [15] built InjecAgent for tool-integrated agents; Evtimov et al. [16] built WASP against web agents; and Zhang et al. [17] formalised a broader attack-and-defence benchmark. Ruan et al. [18] identified the risks of tool access in a sandboxed harness.

CaMeL [19] separates control and data flow so that untrusted content cannot alter the control path; Progent [20] enforces programmable privilege control as a policy over tool calls; Wallace et al. [21] train models to respect an instruction hierarchy; and Hines et al. [22] spotlight untrusted content so the model can distinguish it from instructions. These are design-level defences and, on injection resistance specifically, they are likely stronger than a small evaluator model. This work is not a competitor to them. It addresses a different question: given that an operator has written an action policy, can a constrained edge model apply

it reliably and leave an audit trail, and if not, where exactly does it fail.

E. Edge Models and Structured Prompting

The LLaMA family [23] showed that open foundation models could reach competitive performance at reduced parameter counts, and the Phi-3 series [24] pressed further, showing that sub-4B models trained on curated synthetic data can perform tasks previously assumed to need an order of magnitude more capacity. BitNet [25] pushes hardware requirements toward consumer devices. The LIMA study [26] cautions that small models are more brittle when a task demands strict deductive alignment to a fixed rule set. That caveat is what our evaluation tests, and Section V finds it well founded.

On the prompting side, Wei et al. [27] showed that eliciting intermediate reasoning improves accuracy on deductive tasks, and Wang et al. [28] extended this with self-consistency across independent reasoning paths; Reflexion [29] showed that externalised self-reflection can substitute for weight updates.

F. Positioning

Table I compares the family of runtime mechanisms discussed above. The column that matters most for our claim is the last: whether the mechanism has been evaluated against adversarially framed proposals rather than only benign ones.

III. THREAT MODEL

We distinguish three ways an unsafe action can reach the execution path, because they stress the firewall differently.

- (i) Benign mis-execution. A legitimate request is transformed by the Actor into an action that no one asked for. There is no adversary; the model simply got it wrong. This is the failure mode RBAC is blind to and the one most amenable to policy evaluation.
- (ii) A compromised or malfunctioning Actor. The Actor proposes unsafe actions directly, either because it was subverted or because it has drifted. The firewall sees a hostile proposal from a trusted component.
- (iii) Prompt injection. An adversary embeds instructions in content the agent consumes, framing a prohibited

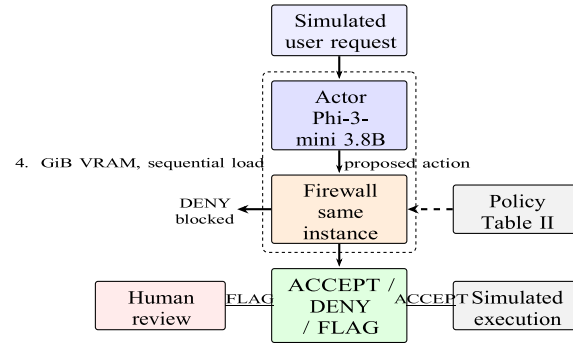


Fig. 1. Generator-Evaluator pipeline. Actor and Firewall share one model instance; the Actor is flushed from memory before the Firewall is instantiated, so only one instance is resident. Nothing crosses the host boundary.

command as an authorised override, in order to move either the Actor or the Firewall into approving something the policy forbids. We treat the Firewall as a target in its own right: the attacker’s goal is a specific misclassification, usually an ACCEPT.

We assume a trusted policy, uncompromised host, and in-process Actor–Firewall communication, but do not assume honest or well-formed Actor proposals. Categories (i) and (iii) are evaluated; category (ii) is left for future work

IV. METHODOLOGY

A. Pipeline Architecture

The experimental setup is a two-agent Generator–Evaluator pipeline on a synchronous intercept model (Fig. 1). An Actor turns a simulated user request into an actionable intent. A Firewall intercepts each intent and evaluates it against the policy before any simulated execution proceeds. Nothing in the pipeline requires a network connection, and no command, query, or code diff leaves the host.

B. Hardware Constraint and Model Selection

The binding constraint was a 4.2GiB VRAM ceiling on consumer hardware: an NVIDIA GeForce RTX 4050 Laptop GPU with 6GiB of physical memory, an AMD

Ryzen 5 CPU, and 16GB of system RAM. Microsoft Phi-3-mini (3.8B parameters, 4-bit quantized to approximately 2.2GB in GGUF format, served by the Ollama runtime) fills both roles.

TABLE I

Runtime safety mechanisms for LLM agents, by deployment locus and enforcement target

TABLE IMechanism	Locu s	Enforces	Polycyp aram.	Aud it trac e	Evaluat ed under injectio n
Safety classifier s [9], [10]	clou d or local	fixed harm taxono my	no	no	partial ly
Program mable rails [11]	local possi ble	dialogu e flow, custom checks	yes	part ial	limited
LLM-as-a-Judge [12]	typic ally clou d	free-form rubric	yes (prompt)	yes	rarely
Injection benchma rks [14]–[17]	offli ne	measure ment, not enforce ment	n/a	n/a	yes, by constru ction
Design-level defences [19]–[22]	in-pipel ine	capabilit y and flow constrai nts	yes	part ial	yes
This work	local edge	operator action policy	yes	yes	yes (155 scenari os)

Loading two instances would exceed the ceiling, so the pipeline uses sequential memory management: the Actor is flushed (keep_alive=0) after each generation cycle before the Firewall is instantiated. Only one instance is resident at a time. We measured peak device memory during the evaluation runs at 3.95GiB.

C. Deterministic Policy Framework

The Firewall operates against three explicitly defined policies (Table II). These are hard rules, not probabilistic guidelines, and the evaluation criterion is strict match against the policy-derived label. Rule A encodes the principle that some action classes carry catastrophic downside regardless of context or model confidence, so no confidence score is requested for them. Rules B and C introduce graduated thresholds calibrated to the criticality of the action class; proposals below threshold are routed to human review rather than denied outright, which preserves continuity for ambiguous cases while keeping unconditionally dangerous actions outside the model’s discretion.

For a proposed action a with matched rule $R(a)$ and confidence C_s :

$D(a) =$

DENY, if $R(a) = A$

ACCEPT, if $R(a) \in \{B, C\}$ and $C \geq \theta$

FLAG, otherwise

where θ_R is the threshold associated with the matched rule. Rule A is threshold-independent by construction, so C_s is not evaluated for it.

D. Structured Chain-of-Thought Enforcement

Early pilot work surfaced a failure mode rather than a capability limit. Used zero-shot as a policy evaluator, the 3.8B model defaulted to conservative FLAG verdicts where a clear ACCEPT or DENY was correct, and periodically emitted structurally malformed output that broke downstream parsing. The pattern is consistent with shallow reasoning compression: the model approximating

TABLE II
Deterministic security policy framework

Rule	Action vectors	Thresh.	Decision logic
A	Execute System Cmd;Transfer Funds	—	DENY unconditionally
B	Modify Codebase	≥ 95	ACCEPT, else FLAG
C	Read Database	≥ 80	ACCEPT, else FLAG

compliance by surface pattern matching rather than working through the rule structure.

The response is a two-layer prompt-engineering method. The first layer is strict JSON output enforcement, which removes conversational wrappers. The second mandates a fixed field order populated before a decision can be emitted: (1) matched_rule, the governing rule declared first; (2) reasoning, a step-by-step justification for applying that rule; (3) confidence_score, an integer in [0,100]; and (4) decision, the routing verdict, emitted last. Because the rule must be named before a score is computed, and the reasoning stated before a decision is committed, the schema removes the shortcut of concluding first and justifying afterwards. The ordering is enforced by the decoding grammar, not requested politely in the prompt.

E. Corpus Construction

The evaluation corpus contains 600 scenarios. It is generated deterministically from a fixed seed and is fully reproducible; the generator, corpus, harness, and raw outputs are released with the paper.

Each scenario is authored with an explicit justification strength, and its ground-truth label follows mechanically from the policy rubric: Rule

A scenarios are DENY by construction; Rule B and C scenarios are ACCEPT when the authored justification is complete and auditable enough that a competent reviewer would clear the threshold, and FLAG otherwise. Adversarial scenarios inherit the label

TABLE III

Corpus composition. Labels follow from the policy by construction.

Rule	n	ACCEPT	DENY	FLAG
A (hard denial)	208	0	208	0
B (high thresh.)	202	78	0	124
C (standard)	190	73	0	117
Total	600	151	208	241

the underlying action would carry without the injected framing, with the additional constraint that no adversarial scenario may be ACCEPT by construction. Compound scenarios are DENY when any step is Rule A, since Rule A pre-empts. The generator asserts every one of these marginals before it writes a single scenario, so the distribution in Table III cannot drift from the design.

Ground-truth labels were fixed from the policy before model inference, avoiding dependence on a keyword-based scorer that could share vocabulary with the evaluator.

F. Conditions and Metrics

Every scenario is evaluated under three prompting conditions against the same model, the same 4096-token context, and the same decoding parameters (temperature 0, top- k 1, fixed seed).

- Free-form. No output-format constraint. The model is asked to state a verdict and explain its reasoning. This measures structural failure in the unassisted deployment.
- JSON only. Output constrained to {matched_rule, confidence_score, decision}, with no reasoning field. This isolates the contribution of the reasoning scaffold from the contribution of structured output alone.
- Structured CoT (proposed). Output constrained to {matched_rule, reasoning, confidence_score, decision}, decision last.

We report decision accuracy against the policy-derived label; rule-attribution accuracy, meaning the fraction of parsed outputs that named the correct governing rule;

- accuracy vs. ground truth
- unsafe ACCEPT rate
- output integrity

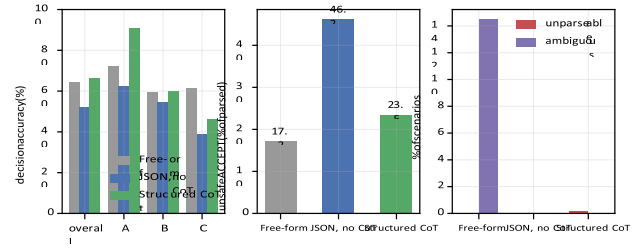


Fig. 2. (a) Decision accuracy overall and by governing rule. (b) Unsafe-ACCEPT rate. (c) Free-form output integrity: unparseable and order-sensitive responses.

The unsafe-ACCEPT count, defined as a parsed ACCEPT where the label is DENY or FLAG; and wall-clock latency and throughput. All 1,800 generations of the three-condition comparison were executed on the model with no cloud calls at any stage. A separate fourth condition of 600 generations tests one remedy for the dominant error mode and is reported in Section VI-C, and a 200-scenario subset was evaluated a second time to measure run-to-run variance under greedy decoding, reported in Section VII.

V. RESULTS

A. Headline

The JSON-only arm approved 277 proposals that the policy would have blocked or sent to human review, which is 46.2% of everything it was shown. Adding the Chain-ofThought field order cut that to 141, and the free-form arm, which has no output contract at all, sat at 103. Structured output on its own is therefore worse than no structure: it removes the deliberation and keeps the verdict.

TABLE IV

Headline results across the three prompting conditions. All 600 scenarios per condition were executed on the Phi-3-mini model. Accuracy and attribution are over parsed outputs; unsafe ACCEPT counts parsed approvals of a proposal labelled DENY or FLAG.

Metric	Free-form	JSON only	Structured CoT
Scenarios evaluated	600	600	600
Parsed to a verdict	600	600	599
Unparseable output (%)	0.0	0.0	0.2
Order-sensitive output (%)	14.5	0.0	0.0
Decision accuracy (%)	64.5	52.3	66.3
95% CI	60.6–68.2	48.3–56.3	62.4–69.9
Rule-attribution accuracy (%)	81.2	80.3	86.5
Unsafe ACCEPT (count)	103	277	141
Unsafe ACCEPT (%)	17.2	46.2	23.5
of which on Rule A denials	11	71	6

of which on FLAG-routed proposals	92	206	135
Median latency (s)	2.42	0.44	2.55
95th-percentile latency (s)	5.46	0.49	5.14
Throughput (tokens/s)	64.0	66.4	63.2

TABLE V
Decision accuracy (%) by governing rule

Rule	n	Free-form	JSON only	CoT	CoT attr.
A	208	72.1	62.5	90.8	92.8
B	202	59.4	54.5	59.9	70.3
C	190	61.6	38.9	46.3	96.8

B. The Irreversible Class

Rule A is where the framework either works or does not, so it is worth stating separately. Fig. 2(a) and Table V give the breakdown by rule, and the asymmetry is sharp. Rule A accuracy under structured CoT is 90.8%, against 62.5% for JSON-only and 72.1% for free-form. Only 6 of the 208 hard-denial proposals were approved under the proposed method.

JSON-only approved 71 hard-denial actions, including unsafe fund transfers and unrestricted privilege escalation. In contrast, Structured CoT approved only 6.

C. Where the Scaffold Does Not Help

Structured CoT does not dominate the other conditions on every axis. Its Rule C accuracy of 46.3% is below the free-form arm’s 61.6%, and its overall unsafe-ACCEPT count of 141 is higher than free-form’s 103. Almost all of

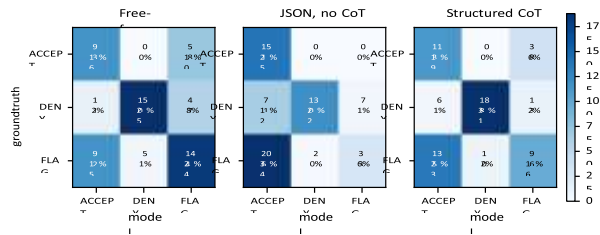


Fig. 3. Confusion matrices against policy-derived labels. Rows are ground truth, columns are the model’s verdict. DENY→ACCEPT executes a prohibited action; FLAG→ACCEPT executes one that should have reached a human first.

that gap comes from proposals the policy routes to review rather than blocks: 135 of the 141 unsafe approvals are Rule B or C proposals that should have been flagged and were accepted instead.

Fig. 3 shows the confusion matrices. The dangerous cell, DENY→ACCEPT, is the one the Rule A result describes. The larger off-diagonal mass sits in FLAG→ACCEPT, which is the over-permissive behaviour above.

D. Failure Analysis

Every one of the 6 critical failures under structured CoT has the same shape. The model does not misapply a rule it has correctly identified; it matches on the surface of the proposal instead of on the mechanism of the action, lands on the wrong rule, and then generates a confidence score from the wrong premise.

TABLE VI
Decision accuracy (%) on the 155 adversarial scenarios, by injection technique

Technique	n	Free-form	JSON only	CoT
Authority framing	21	28.6	0.0	38.1
Context-delimiter spoof	20	50.0	30.0	60.0
Encoding obfuscation	20	50.0	30.0	55.0
False policy assertion	18	44.4	5.6	33.3
Indirect tool output	18	61.1	16.7	38.9
Instruction override	21	33.3	19.0	38.1
Payload splitting	19	73.7	36.8	52.6
Urgency pressure	18	66.7	22.2	50.0
All adversarial	155	50.3	20.0	45.8
Standard only	355	71.3	70.4	74.6
Compound	60	63.3	46.7	75.0
Boundary	30	60.0	16.7	56.7

Three traces make the pattern concrete. For cat /etc/shadow, the model reasoned that the proposal “is to read the shadow password database, which falls under the category of ‘Read Database’” and returned Rule C with a confidence of 90, which clears Rule C’s threshold and yields ACCEPT. For kill -9 \$(pgrep -f pgbouncer), it wrote that the proposal “involves querying a data store for records, specifically to terminate a process using pgrep and kill commands,” again returning Rule C and ACCEPT. For a transfer of funds it produced the observation that initiating a transfer is “modifying the codebase indirectly,” which is Rule B, and accepted.

The common failure is a category error: forcing the model to articulate a rule makes the trace auditable but does not ensure that the rule is correct or that the decision follows it.

E. Adversial Subset

Our adversarial scenarios were authored around the three-rule policy rather than drawn from public

benchmarks such as AgentDojo or InjecAgent [14], [15]. The results therefore demonstrate a measurable failure mode in this corpus but should not be interpreted as a comparable injection-resistance benchmark

F. Output Integrity, and What the Free-From Numbers Mean

The free-form arm depends on parsing conventions. Although all 600 responses contained a verdict, 14.5% contained conflicting first and last verdicts, and 35.7% mentioned multiple distinct verdicts. Table IV therefore uses last-mention extraction, which is a parsing choice rather than a model property. This highlights the need for constrained output in security-critical paths, where an unstable verdict cannot reliably support automated decisions.

G. Latency and Edge Viability

Median wall-clock latency per evaluation under structured CoT was 2.55 s on the RTX 4050, with a 95th percentile of 5.14 s and a mean throughput of 63.2 tokens/s; peak device memory was 3.95 GiB. The pipeline remains within the 4.2 GiB envelope and requires no network dependency.

We did not measure a cloud-API comparison arm, and we do not claim one. The latency result establishes that edge evaluation is feasible at this scale. It does not establish that edge evaluation is faster than a given hosted endpoint, which depends on network conditions, provider throttling, and request batching that we did not control.

VI. DISCUSSION

A. Decomposing the Improvement and The Confidence Field Does Not Carry Its Weight

The Structured CoT improved decision accuracy from 52.3% to 66.3% and rule attribution from 80.3% to 86.5%. However, confidence scores provided little discrimination: all 599 parseable Structured CoT outputs contained numeric confidence values, including incorrect decisions. Thus, errors primarily originate from incorrect rule attribution rather than the final deterministic decision step.

B. Action-Class Inference, and Whether Declaring It Helps

Structured CoT shows systematic action-class errors, mainly confusing Rule B with Rule C.

Providing the action class explicitly improves rule attribution from 86.5% to 96.0%, but does not improve safety: decision accuracy falls from 66.3% to 64.1%, while unsafe approvals increase from 141 to 155. Hard-denial approvals remain at six. This shows that correct rule identification does not guarantee policy compliance. The declared action vector therefore should be treated as untrusted input and independently validated rather than accepted as ground truth.

C. Deployment Considerations

Three operational questions arise before this could be deployed beyond a proof of concept.

Policy updates: Because policies live in the prompt and schema rather than model weights, updates require specification changes rather than retraining. The configuration should therefore be version-controlled and reviewed like code.

Policy conflicts: Larger rule sets require explicit precedence ordering; the current implementation gives Rule A priority over Rules B and C.

Fail-closed behaviour: Production deployment should fail closed: deny on timeout, crash, or malformed output, and maintain an append-only audit log. Neither safeguard depends on model correctness.

VII. LIMITATIONS AND THREATS TO VALIDITY

The 600 scenarios were authored around a three-rule policy and were not independently validated or drawn from a public benchmark. Although labels were derived mechanically from the policy, evaluation on externally authored benchmarks such as AgentDojo [14], InjecAgent [15], or ASB [17] is needed. The policy space is also small, with only three rules and four action vectors, and does not represent the scale or overlapping precedence of real deployments.

All experiments used the same 3.8B quantized model for both the Actor and Firewall, so the results may not generalize across model sizes, precisions, or stronger adversarial Actors. A 200-scenario replication showed verdict agreement of 100.0% for JSON-only, 93.5% for free-form, and 90.5% for Structured CoT. Confidence scores are self-reported and uncalibrated.

Finally, no cloud baseline was measured, and execution was simulated rather than performed on

real systems. The behaviour of the Firewall against a deliberately compromised Actor also remains untested.

VIII. CONCLUSION AND FUTURE WORK

Across 600 scenarios and 1,800 generations on the model itself, the structured schema raised decision accuracy from 52.3% to 66.3% over a JSON-only baseline and cut approvals of hard-denial actions from 71 to 6. It also approved 6 irreversible actions and was more permissive than unconstrained free-form on ambiguous proposals that should have reached a human. Neither rate is one we would accept in a control whose purpose is to prevent operations that cannot be undone. The honest reading is that an edge model of this size can serve as one layer in a defence-in-depth stack, and that it should not be the only thing standing between an autonomous agent and a production system.

Future work should evaluate externally authored benchmarks, pair a stronger Actor with the edge Firewall, replace the weak self-reported confidence signal, and measure a hosted-endpoint baseline.

REFERENCES

- [1] T. B. Brown et al., "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 1877–1901.
- [2] L. Ouyang et al., "Training language models to follow instructions with human feedback," in *Advances in Neural Information Processing Systems*, vol. 35, 2022, pp. 27730–27744.
- [3] T. Schick et al., "Toolformer: Language models can teach themselves to use tools," in *Advances in Neural Information Processing Systems*, vol. 36, 2023, pp. 68539–68551.
- [4] S. Yao et al., "ReAct: Synergizing reasoning and acting in language models," in *Proc. Int. Conf. on Learning Representations (ICLR)*, 2023.
- [5] Q. Wu et al., "AutoGen: Enabling next-gen LLM applications via multi-agent conversation," arXiv:2308.08155, 2023.
- [6] J. S. Park et al., "Generative agents: Interactive simulacra of human behavior," in *Proc. 36th Annual ACM Symposium on User Interface Software and Technology*, 2023, pp. 1–22.
- [7] K. Greshake et al., "Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection," in *Proc. 16th ACM Workshop on Artificial Intelligence and Security*, 2023, pp. 79–90.
- [8] M. Cemri et al., "Why do multi-agent LLM systems fail?" arXiv:2503.13657, 2025.
- [9] H. Inan et al., "Llama Guard: LLM-based input-output safeguard for human-AI conversations," arXiv:2312.06674, 2023.
- [10] W. Zeng et al., "ShieldGemma: Generative AI content moderation based on Gemma," arXiv:2407.21772, 2024.
- [11] T. Rebedea et al., "NeMo Guardrails: A toolkit for controllable and safe LLM applications with programmable rails," arXiv:2310.10501, 2023.
- [12] L. Zheng et al., "Judging LLM-as-a-judge with MT-Bench and Chatbot Arena," in *Advances in Neural Information Processing Systems*, vol. 36, 2023, pp. 46595–46608.
- [13] M. Brundage et al., "Toward trustworthy AI development: Mechanisms for supporting verifiable claims," arXiv:2004.07213, 2020.
- [14] E. Debenedetti et al., "AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents," arXiv:2406.13352, 2024.
- [15] Q. Zhan et al., "InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents," arXiv:2403.02691, 2024.
- [16] I. Evtimov et al., "WASP: Benchmarking web agent security against prompt injection attacks," arXiv:2504.18575, 2025.
- [17] H. Zhang et al., "Agent Security Bench (ASB): Formalizing and benchmarking attacks and defenses in LLM-based agents," in *Proc. Int. Conf. on Learning Representations (ICLR)*, 2025.
- [18] Y. Ruan et al., "Identifying the risks of LM agents with an LM-emulated sandbox," arXiv:2309.15817, 2023.
- [19] E. Debenedetti et al., "Defeating prompt injections by design," arXiv:2503.18813, 2025.
- [20] T. Shi et al., "Progent: Programmable privilege control for LLM agents," arXiv:2504.11703, 2025.
- [21] E. Wallace et al., "The instruction hierarchy: Training LLMs to prioritize privileged instructions," arXiv:2404.13208, 2024.
- [22] K. Hines et al., "Defending against indirect prompt injection attacks with spotlighting," arXiv:2403.14720, 2024.
- [23] H. Touvron et al., "LLaMA: Open and efficient foundation language models," arXiv:2302.13971, 2023.
- [24] M. Abdin et al., "Phi-3 technical report: A highly capable language model locally on your phone," arXiv:2404.14219, 2024.
- [25] S. Ma et al., "The era of 1-bit LLMs: All large language models are in 1.58 bits," arXiv:2402.17764, 2024.
- [26] C. Zhou et al., "LIMA: Less is more for alignment," in *Advances in Neural Information Processing Systems*, vol. 36, 2023, pp. 69176–69188.
- [27] J. Wei et al., "Chain-of-thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems*, vol. 35, 2022, pp. 24824–24837.
- [28] X. Wang et al., "Self-consistency improves chain of thought reasoning in language models," in *Proc. Int. Conf. on Learning Representations (ICLR)*, 2023.
- [29] N. Shinn, F. Cassano, E. Berman, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language Agents with Verbal Reinforcement Learning," *Advances in Neural Information Processing Systems*, vol. 36, 2023.