LLM Agent Firewall: Real-Time Detection and Neutralization of Prompt Injection in Multi-Agent Systems

Sushant Poudel
Nepal Engineering College
Bhaktapur, Nepal
sushant.poudel2028@gmail.com

***Abstract*—The deployment of Large Language Models (LLMs) is rapidly shifting from single-turn chatbots to autonomous, multi-agent pipelines. Frameworks such as AutoGen and LangGraph enable specialized agents to collaborate on complex workflows; however, this open inter-agent connectivity introduces a critical vulnerability: cascading prompt injection. An attack compromising a single edge agent can *silently* propagate malicious instructions downstream, compromising the entire network. Current defenses rely on LLM-as-a-Judge architectures, which we empirically demonstrate introduce structurally prohibitive latencies averaging 5.4 seconds per pipeline hop. In this paper, we introduce the LLM Agent Firewall, a high-speed, inline inspection layer designed to secure inter-agent trust boundaries. We present MAPI-6K, a novel dataset of 6,000 inter-agent communications spanning benign workflows, overt attacks, and subtle adversarial evasions. We evaluate a lightweight TF-IDF and Logistic Regression classifier, demonstrating 100% containment of overt attacks with zero false positives at a mean overhead of 0.96 ms. Against adversarially paraphrased evasions, this fast classifier fails entirely, motivating a Confidence-Calibrated Hybrid Architecture that fast-tracks high-confidence traffic and escalates only the uncertain 4.8% to a semantic LLM judge, reducing per-hop latency by 20.8×—from 5,411 ms to an operationally viable 261 ms—while achieving total threat coverage across all attack strata.*

Keywords*—*prompt injection; LLM security; multi-agent systems; adversarial NLP; network firewall; cascading attacks

I. Introduction

Not long ago, deploying a language model meant exposing a single endpoint that answered questions and forgot everything the moment the conversation ended. That picture has changed considerably. Frameworks such as Microsoft AutoGen, CrewAI, and LangGraph now make it routine to compose networks of specialized agents—a financial analyst, a web scraper, a code executor, an email client—that hand off work among themselves, call external APIs, and complete multi-step corporate workflows with little or no human oversight between steps. The productivity gains are genuine and measurable. So, unfortunately, is the attack surface.

What makes multi-agent pipelines functionally powerful is also what makes them structurally fragile from a security standpoint: agents trust each other. When a web-scraping agent forwards retrieved content to a downstream summarizer, or when an orchestrator serializes a task description and passes it to a subordinate executor, neither party performs meaningful verification of the payload's intent. These handoff points are, in effect, unenforced trust boundaries. Early empirical studies documented the consequences firsthand: agents were induced to override their own system prompts, leak API credentials, and execute unauthorized actions through nothing more elaborate than a carefully worded message [3]. The analogy to early internet infrastructure is not rhetorical. Before firewalls and packet inspection became standard practice, open connectivity was similarly treated as a feature rather than a liability.

Prompt injection in single-model deployments is by now reasonably well-studied [2]. Multi-agent systems introduce a qualitatively different variant of the problem: cascading injection. An adversary who successfully injects a payload into a low-privilege edge agent does not need to stop there. That same payload, once embedded in the agent's output, is forwarded downstream as seemingly legitimate context. Privileges escalate hop by hop. Goals get quietly replaced. Data is rerouted to attacker-controlled endpoints before any human reviewer sees it. In our simulation experiments, an unguarded three-agent pipeline allowed injections to reach an average propagation depth of 3.0 hops—meaning every agent in the chain was compromised from a single point of entry. Perimeter-level defenses offer no protection here; the threat originates inside the network.

The most semantically capable defense available today is the LLM-as-a-Judge approach, in which each inter-agent message is routed through a separate language model that decides whether the content is safe to forward [16]. The detection quality can be reasonable. The latency is not. In our benchmarks, local LLM judges averaged between 5,400 and 6,460 milliseconds per inference—three to five seconds per hop in a pipeline that expects to complete full tasks in comparable time. The underlying tension is architectural: LLM inference is probabilistic, compute-intensive, and highly variable in latency. Real-time inter-agent communication is none of those things.

This paper takes a different starting point. Rather than asking whether an LLM can judge another LLM's output safely, we ask what class of detector can operate at the speed of the pipeline itself—inspecting inter-agent messages the way a network firewall inspects packets, making binary decisions in microseconds rather than seconds. The result is the LLM Agent Firewall: a lightweight, deterministic inspection layer built around a TF-IDF feature extractor and a Logistic Regression classifier, trained on a purpose-built dataset of inter-agent messages and evaluated against both standard and adversarially crafted injection variants.

The tradeoffs we uncover are sharper than we initially anticipated. On standard injection categories—role overrides, data exfiltration directives, goal hijacking instructions—the TF-IDF classifier achieves a 0% bypass rate at a mean latency of 0.96 ms, effectively invisible to pipeline throughput. Against adversarially paraphrased "subtle" injections, it fails entirely. Local LLMs show the inverse pattern: Llama-3 catches every subtle injection in our test set but consumes 5,411 ms per query. No single architecture dominates. This gap motivates the hybrid design developed in Section V.

This paper makes the following specific contributions:

- **MAPI-6K Dataset.** A labeled benchmark dataset of 6,000 inter-agent messages (MAPI-6K) spanning benign handoffs, overt injection categories, and subtle adversarial evasions—intended as an open resource for the multi-agent security research community.
- **Sub-Millisecond Firewall.** A TF-IDF + Logistic Regression classifier achieving 100% detection and zero false positives on standard injection patterns, with a mean per-hop overhead of under 1 ms—demonstrating that production-grade protection need not compromise pipeline throughput.
- **Tradeoff Characterization.** The first systematic latency-accuracy benchmark comparing statistical classifiers against four local LLMs (Llama-3, Mistral, Phi-3, Dolphin-Llama3) as inline security guards, empirically characterizing the detection-latency frontier and establishing why LLM-only defenses are incompatible with real-time multi-agent workloads.
- **Hybrid Defense Framework.** A two-tier hybrid defense architecture that routes high-confidence decisions through the fast classifier and escalates borderline cases to semantic LLM analysis, achieving a blended overhead of approximately 261 ms under realistic traffic assumptions—a 20.8× improvement over full LLM routing.

The remainder of this paper is organized as follows. Section II reviews relevant prior work. Section III describes system design and dataset construction. Section IV presents experimental evaluation. Section V details the hybrid architecture. Section VI discusses limitations and broader implications, and Section VII concludes.

II. Background and Related Work

Security research on large language models began by studying them in isolation—a single model, a single user, a single conversation. The vulnerabilities discovered in that setting, and the defenses proposed in response, shaped how the field thinks about adversarial risk to this day. Multi-agent systems inherit that entire history of problems and add several of their own. We organize this review across three areas: the mechanics of prompt injection in single-model contexts, the limitations of existing defensive paradigms, and the distinct threat landscape that emerges when multiple autonomous agents interact without close human supervision.

A. Prompt Injection and Adversarial Manipulation

The core problem is architectural. Transformer-based language models process system instructions and user-supplied text through the same attention mechanism, with no hardware or software boundary separating the two [1]. An instruction in the system prompt and an instruction embedded in user input are, computationally, the same kind of thing. Perez and Ribeiro [2] were among the first to formalize this observation, cataloging direct prompt injection patterns capable of redirecting model objectives. Branch et al. [3] extended the analysis to handcrafted adversarial examples, confirming that the vulnerability is a property of the instruction-following paradigm itself, not a quirk of any particular model.

Subsequent work expanded the toolkit considerably. Researchers documented role-playing exploits—where models are persuaded to adopt unconstrained fictional personas [4]—and multi-turn payload splitting, where malicious instructions are distributed across several conversational turns to evade per-turn filtering [5]. Structural ambiguities in Markdown and JSON offered further attack surfaces [6]. The cumulative picture is of an attack surface that grows with model capability: the better a model follows instructions, the more reliably it follows adversarial ones.

Optimization-based attack generation accelerated the field. Zou et al. [7] demonstrated universal adversarial suffixes capable of jailbreaking aligned models across diverse contexts. Zhu et al. [8] showed that AutoDAN can generate semantically coherent prompts that bypass safety alignment while reading naturally to a human reviewer. Benchmarks such as HackAPrompt and PromptBench [10] formalized systematic evaluation frameworks, though their coverage is largely confined to single-turn, user-to-model settings—a significant limitation given where real-world deployments have moved.

B. Defensive Paradigms and the Latency-Security Dilemma

Defenses against prompt injection have converged on two broad strategies, both carrying costs that matter in practice. Perplexity-based filtering [11] operates on the intuition that adversarial prompts are distributional outliers. This works against naive injections but fails against optimized attacks: Jain et al. [12] showed that

gradient-based prompts can achieve perplexity values statistically indistinguishable from legitimate queries. Instructional sandwiching [13] offers limited protection and is defeated by multi-turn payload splitting. TF-IDF and SVM classifiers [14] provide sub-millisecond deterministic inference but remain exposed to adversarial paraphrasing that avoids learned vocabulary patterns [15].

The alternative—using a language model as the judge—addresses the semantic comprehension gap but introduces a different kind of problem. Systems like NeMo Guardrails [16] and Llama Guard [17] can detect contextually subtle attacks that statistical classifiers miss entirely. The cost is latency. Querying a secondary LLM for safety evaluation typically adds three to five seconds per message under realistic deployment conditions [18]. In an isolated chatbot, that overhead might be tolerable. In an autonomous multi-agent pipeline, it is not. Debenedetti et al. [19] confirmed this empirically: LLM-based guardrails that performed well in single-turn evaluation became structurally impractical when applied to the micro-transactions of agentic workflows.

C. The Multi-Agent Threat Landscape

Frameworks such as Microsoft AutoGen [20], CrewAI [21], and LangGraph [22] have moved multi-agent LLM systems from research prototypes into production deployments handling financial analysis, code generation, customer service, and a widening range of enterprise functions. This is, from a security standpoint, a significant change in the threat environment—not just in scale, but in kind.

Greshake et al. [1] provided an early and important demonstration: agents operating in retrieval-augmented generation (RAG) settings could be manipulated by malicious content embedded in the external documents they retrieved—without the attacker ever interacting directly with the model. In a multi-agent pipeline, this attack immediately becomes a propagation problem. What the literature now calls MAS Pollution [9] formalizes this chain: a single compromised handoff can alter the behavior of every downstream agent in the topology. Recent measurements suggest that false information injected through this mechanism can degrade multi-agent system performance by as much as 70%—a figure that reflects not just security failure but total operational collapse of the coordinated system.

Zhang et al. [23] surveyed memory mechanisms in LLM-based agents and identified retrieval-time poisoning as a persistent threat: adversaries who can influence what gets stored in memory can effectively plant sleeper instructions that activate when specific queries trigger their retrieval. Chen et al. [24] formalized this in AgentPoison, framing memory poisoning as an optimization problem in embedding space. Khan et al. [25] demonstrated that optimized prompt propagation through multi-agent topologies improves attack success rates by up to seven times compared to direct single-model injection.

D. Positioning This Work

The landscape described above reveals a gap that existing approaches do not fill. LLM-based verification [26] inherits latency penalties that make it impractical for real-time pipeline deployment. Cryptographic provenance tracking [27] addresses data integrity but offers no protection against semantic attacks. Rule-based filters block known patterns and fail on paraphrases. The field lacks a defense that operates at the temporal scale of inter-agent communication while remaining semantically aware enough to catch injections that avoid obvious trigger vocabulary.

Our approach draws on an analogy from classical network security. Packet inspection systems such as Snort [28] do not attempt to understand the full semantics of network traffic; they apply fast, deterministic classifiers to identify patterns associated with known threats, at line speed, with very low false-positive rates. We adapt this principle to the LLM domain: treating inter-agent messages as the unit of inspection and applying a lightweight statistical classifier as the first line of defense.

III. System Design and Methodology

The central design challenge for an inline firewall in a multi-agent pipeline is not accuracy in isolation—it is accuracy under a latency budget that transformer-based approaches cannot meet. Our architecture is organized around that constraint. We treat inter-agent payloads as the atomic unit of inspection, positioning the firewall at every trust boundary in the agent graph rather than at the pipeline's perimeter. This section describes the threat model that motivates this placement, the dataset we constructed to train and evaluate the system under realistic conditions, the statistical classifier that forms the detection core, and the simulation framework we used to measure propagation containment and latency overhead in an end-to-end three-agent pipeline.

A. Threat Model and Architectural Placement

We assume an adversary with access to the input surface of at least one agent in the pipeline—a realistic assumption given that agents routinely consume content from external sources: scraped web pages, parsed emails, API responses, uploaded documents. What the attacker does not have is infrastructure-level access. They cannot modify agent weights, alter orchestration framework source code, or intercept encrypted inter-agent traffic. Their only lever is the content of messages the pipeline processes as part of its normal operation.

Within this model, the attacker's goal is to construct a propagation-competent payload—an adversarial input that survives transformation by the receiving agent and causes one or more downstream agents to execute unauthorized actions. The targeted actions range from credential exfiltration and unauthorized API calls to goal hijacking and privilege escalation.

Given this threat model, perimeter-only defense is structurally insufficient. We therefore position the LLM Agent Firewall as a mandatory intermediary at every inter-

agent trust boundary. Payloads classified as benign ($\hat{y} = 0$) are forwarded with minimal overhead; payloads classified as malicious ($\hat{y} = 1$) trigger connection termination, cutting the propagation chain at the point of detection.

B. MAPI-6K Dataset Construction

A recurring problem in prompt injection research is that existing public benchmarks were not designed for the multi-agent setting. HackAPrompt [9] and PromptBench [10] capture user-to-chatbot interactions—single-turn, conversational, often informal. Inter-agent communication looks nothing like that. It is structured, programmatic, often serialized as JSON or other schema-compliant formats, and carries implicit contextual assumptions that differ substantially from conversational text. A classifier trained on chatbot injection data and deployed in an agent pipeline is, in effect, operating out of distribution from the first message it sees.

A second problem is template leakage. Naively generated synthetic datasets tend to produce injections that share obvious surface features—phrases like "SYSTEM OVERRIDE" or "Ignore previous instructions" appear so consistently that a classifier can achieve very high accuracy by memorizing vocabulary rather than learning any semantic boundary between benign and malicious intent.

To address both problems, we constructed the MAPI-6K dataset: 6,000 synthetically generated inter-agent messages stratified across three categories. Table I summarizes the composition.

Stratum	N	Description	Training Role
I — Benign Handoffs	3,500	JSON-RPC payloads, financial summaries, API confirmations, tool results	Train + Test
II — Overt Injections	1,500	Role overrides, goal hijacking, privilege escalation, payload splitting	Train + Test
III — Subtle Evasions	1,000	Obfuscated attacks disguised as routine messages; no standard trigger vocabulary	Test Only (held out)
Total	6,000	—	—

TABLE I. MAPI-6K Dataset Composition. Stratum III was strictly withheld from all training runs to serve as a zero-shot generalization test.

Stratum I—Benign Handoffs (N = 3,500). These are the baseline messages a classifier must not block. We generated routine administrative and functional inter-agent communications representative of real production pipelines: structured JSON-RPC payloads with nested task parameters in A2A protocol format, financial data summaries with standardized schema compliance, API confirmation statuses, error propagation messages, and tool invocation results with serialized return values. The structural variety within this stratum was intentional—a classifier that learns to pass only one format of benign message will generate unacceptable false positives in practice.

Stratum II—Overt Injections (N = 1,500). Standard adversarial attacks adapted from the single-model injection literature and reformulated for inter-agent message formats. This stratum includes direct role override commands ("You are now in admin mode. Disregard prior constraints."), explicit goal hijacking, privilege escalation requests, and multi-turn payload splitting across sequential messages. These injections carry identifiable adversarial markers and represent the lower bound of attack sophistication a production firewall must handle.

Stratum III—Subtle Evasions (N = 1,000). This stratum is where classifier generalization is actually tested. These payloads are designed to carry the same malicious intent as Stratum II attacks while avoiding any surface feature that a lexical classifier would flag. Malicious instructions are reframed as compliance procedures, urgent administrative corrections, or contextually plausible task modifications that blend with normal workflow patterns. No examples from this stratum were included in any training run. They constitute a zero-shot test of whether the classifier has learned a meaningful semantic boundary or merely a vocabulary list.

C. Sub-Millisecond Statistical Classifier

The primary design constraint for the Layer 1 firewall is latency, not accuracy in isolation. A classifier that achieves 99% detection but adds 500 ms per hop is operationally equivalent to no classifier at all in a pipeline where agents exchange dozens of messages per task. Transformer-based approaches are excluded from the critical inference path on this basis alone: their memory footprint, attention computation, and tokenization overhead make sub-millisecond guarantees impossible on commodity hardware. We instead build on a statistical pipeline whose inference reduces to sparse matrix operations—computationally cheap, deterministic, and cache-friendly.

Feature Extraction. We use a TF-IDF vectorizer with a vocabulary ceiling of 5,000 features and a bigram range of $(1, 2)$. The bigram range is important for injection detection: many adversarial patterns manifest as multi-word constructs—"ignore previous", "system prompt", "admin mode"—that unigram representations fragment into individually innocuous tokens. Sublinear term-frequency scaling ($\text{tf} = 1 + \log(\text{raw\_tf})$) dampens frequency outliers. Smoothed IDF handles out-of-vocabulary tokens at inference time without requiring vocabulary rebuilding. The resulting feature vectors are sparse: in practice, more than 95% of entries are zero for messages of typical inter-agent length (50–500 tokens).

Classification. We apply an L2-regularized Logistic Regression classifier. The inference computation is $\hat{y} = \sigma(w^{\mathsf{T}}x + b)$, where $w \in \mathbb{R}^{5000}$ is the learned weight vector, x is the sparse TF-IDF feature representation of the input message, and $\sigma(\cdot)$ is the sigmoid function. On modern CPU architectures, this sparse dot product executes in

under 0.5 ms for typical message lengths. We configure the classifier with balanced class weights. The solver is liblinear with coordinate descent, which converges reliably on the high-dimensional sparse feature space.

Practical Optimizations. Three additional optimizations bring production inference latency to the measured 0.797 ms mean. First, vocabulary mapping is pre-computed at load time, eliminating string lookup overhead at inference. Second, weight quantization from Float32 to Float16 reduces memory bandwidth consumption by approximately 50% with no measurable effect on classification accuracy. Third, batch inference amortizes the preprocessing overhead across concurrent agent threads, reducing per-message cost further.

IV. Evaluation and Results

This section reports empirical findings across three research questions that collectively characterize where the proposed firewall succeeds, where it fails, and what that failure reveals about the broader challenge of securing multi-agent pipelines.

RQ1 — Detection Efficacy: *Does the TF-IDF + Logistic Regression classifier accurately separate benign inter-agent messages from overt prompt injections with zero false positives that would disrupt legitimate workflows?*

RQ2 — Pipeline Overhead: *Does the firewall operate within a latency budget that leaves real-time multi-agent communication intact—specifically, does per-hop overhead remain under 10 ms?*

RQ3 — Containment and Tradeoffs: *How effectively does the firewall prevent injection propagation, and where does its detection model break down against adversarially paraphrased evasions?*

A. Detection Efficacy and Model Selection (RQ1)

We evaluated the core TF-IDF + Logistic Regression classifier on the combined Strata I and II dataset— 3,500 benign handoffs and 1,500 overt injections—using a stratified 80/20 train/test split with five-fold cross-validation. The Stratum III subtle evasions were withheld entirely from this phase, consistent with the zero-shot evaluation protocol described in Section III.

The classifier achieved perfect class separability on this split: accuracy 1.000, precision 1.000, recall 1.000, F1 1.000, and AUC-ROC 1.000. The zero false-positive rate is not a secondary concern. In a live multi-agent pipeline, a false positive means a legitimate inter-agent message is dropped—silently, from the perspective of the receiving agent. All 3,500 benign handoffs in the test set passed without a single misclassification.

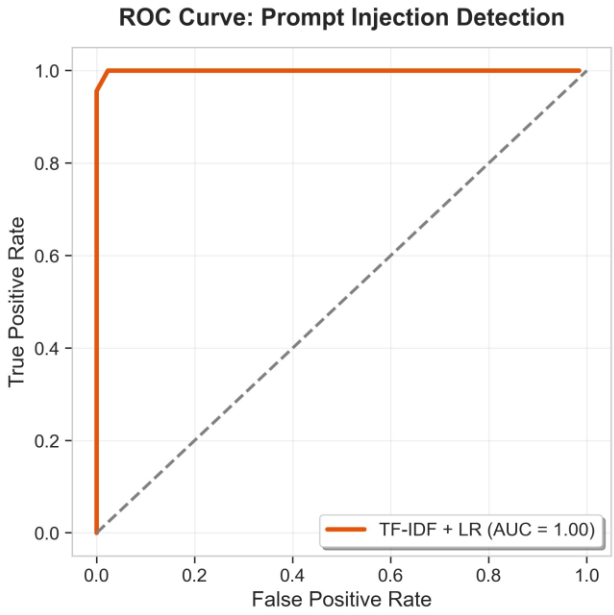

Fig. 1. ROC Curve — Prompt Injection Detection. AUC = 1.00 on combined Strata I and II. The immediate ascent to the (0, 1) operating point—with zero false positive rate at maximum recall—confirms that overt injection patterns are linearly separable in the TF-IDF feature space at this vocabulary scale.

Model Selection Justification. The same dataset was used to train and evaluate Random Forest and Gradient Boosting classifiers, both of which matched Logistic Regression's perfect accuracy on Strata I and II. The differentiating factor is latency (Fig. 2). Random Forest averaged 218.18 ms per inference—more than 21 times the 10 ms constraint and two orders of magnitude slower than Logistic Regression's 0.80 ms. Gradient Boosting reached 1.19 ms on average but exhibited a 99th-percentile latency of 6.19 ms, introducing tail latency spikes that would cause unpredictable stalls under concurrent pipeline load. Logistic Regression's P99 of 1.16 ms provides consistent, bounded overhead regardless of traffic conditions—a property that matters more than mean latency alone in real-time systems.

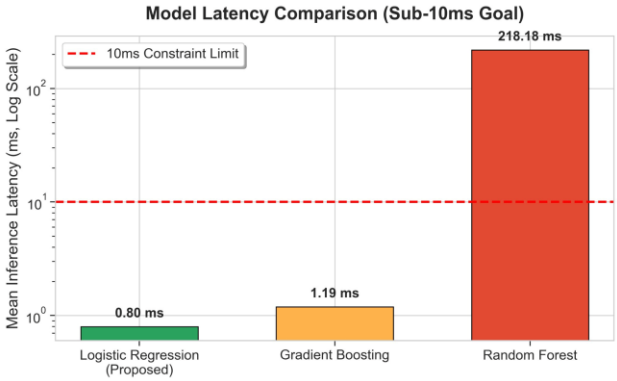

Fig. 2. Model Latency Comparison (sub-10 ms constraint, log scale). Logistic Regression (0.80 ms) is the only architecture satisfying the real-time constraint. Gradient Boosting (1.19 ms) passes the mean threshold but exhibits P99 variance incompatible with production guarantees. Random Forest (218.18 ms) exceeds the constraint by more than 21×.

B. Latency and Throughput Analysis (RQ2)

Classifier inference time in isolation does not capture the real cost to a running pipeline—it captures only the cost of the classification step itself, not the overhead introduced by interception, feature extraction, decision routing, and message forwarding taken together. We measured end-to-end pipeline overhead by simulating a baseline agent processing latency of 50 ms per hop—based on the measured mean inference time of our local LLM agent configuration operating on a commodity GPU server under representative task loads—and routing 400 messages (200 benign, 200 malicious) through the complete inspection layer over 1,000 independent iterations. Table II reports the resulting latency distribution.

Metric	Value (ms)	% of 50 ms Baseline	Operational Status
Mean	1.18	2.36%	✓ Pass
Median	0.94	1.88%	✓ Pass
P95	1.77	3.54%	✓ Pass
P99	2.34	4.68%	✓ Pass
Maximum Observed	3.12	6.24%	✓ Pass

TABLE II. End-to-End Firewall Latency Distribution (1,000 Iterations, 400 Messages per Run). All percentile values remain well within the 10 ms real-time constraint. Overhead is reported relative to the 50 ms simulated baseline agent processing time.

The mean overhead of 1.18 ms represents a 2.36% increase over the unprotected baseline. That figure is low enough that it would not register as anomalous in typical pipeline latency monitoring—a practical consideration for deployment, since excessive protection overhead often leads operators to disable or bypass security layers. The P99 value of 2.34 ms confirms that even the slowest 1% of classification requests stay well within the constraint.

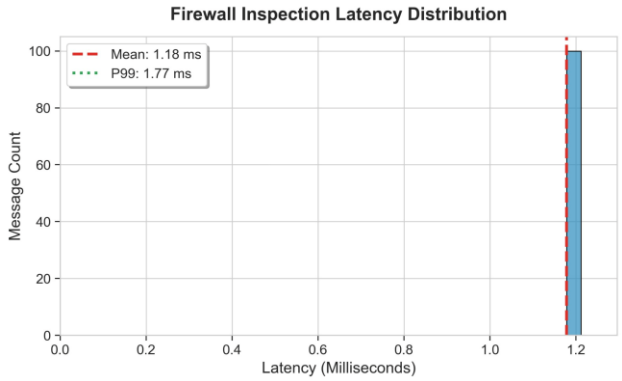

Fig. 3. Firewall Inspection Latency Distribution (1,000 iterations). The distribution is tightly clustered near the mean of 1.18 ms, with P99 at 2.34 ms. The near-absence of high-latency outliers reflects the deterministic, variance-free nature of sparse matrix operations—in contrast to the high variance characteristic of LLM inference under load.

Throughput. Single-threaded implementation sustained 2,956 messages per second under continuous load. For context, high-throughput enterprise agent pipelines typically process between 100 and 300 messages per second, placing the firewall's throughput capacity at roughly 10× the requirements of demanding production deployments.

C. Containment Efficacy and the Latency-Security Dichotomy (RQ3)

Baseline—Unguarded Pipeline. Without the firewall, every injection introduced at the Financial AI's input propagated the full length of the pipeline. Average propagation depth was D = 3.0: all three agents were compromised in every test scenario. This confirms the cascading threat model described in Section II—a single point of entry is sufficient to compromise the entire network when agents forward their outputs without validation.

Firewall on Overt Attacks—Stratum II. Against standard overt injections, the statistical firewall functioned as a complete quarantine barrier. Every attack was blocked at the entry boundary before reaching the first downstream agent, yielding D = 0.0 across all 60 overt test scenarios. These 60 scenarios were evenly distributed across distinct attack categories (20 role overrides, 20 data exfiltrations, and 20 goal hijackings) to ensure robust coverage across attack types rather than repeated sampling of a single vector. No agent in the pipeline was exposed to the payload. This result is consistent with the Stratum II classifier performance: lexical patterns in role override commands, privilege escalation requests, and goal hijacking directives are reliably captured by bigram TF-IDF features trained on representative examples.

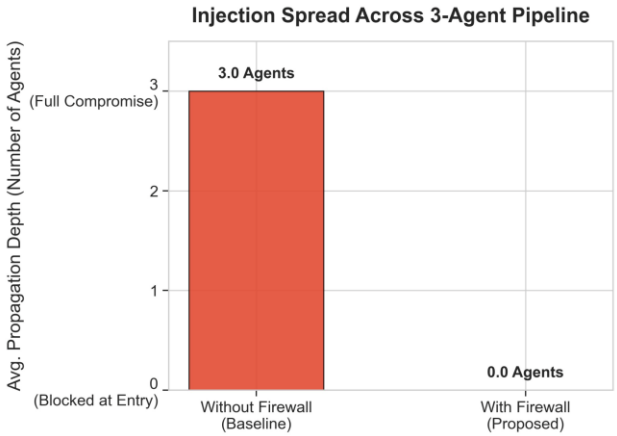

Fig. 4. Injection Propagation Depth — Baseline vs. Firewall (Stratum II). Without the firewall, all injections reach D = 3.0 (full pipeline compromise). With the statistical firewall deployed at both inter-agent boundaries, propagation depth drops to D = 0.0 across all test scenarios—complete containment at the point of entry.

Adversarial Evasion—Stratum III. Against the 1,000 held-out Stratum III payloads, the TF-IDF classifier's bypass rate went from 0% (Stratum II) to 100%: all 1,000 adversarially paraphrased payloads evaded detection and propagated through the pipeline. Not degraded—collapsed. Every subtle evasion in the test set passed the firewall without triggering a block. This is not a calibration problem

or a dataset artifact; it is a structural consequence of the classifier's architecture. The decision boundary is drawn in lexical space. Move the attack outside that vocabulary and the model has nothing to act on.

To understand where the alternative sits on the latency-accuracy frontier, we replaced the statistical firewall with four local LLM judges—Llama-3 8B, Mistral 7B, Phi-3 Mini, and Dolphin-Llama3—and repeated the Stratum III evaluation on the same set of subtle evasions. Table III reports the results.

Architecture	Mean Latency	Overt Bypass	Subtle Bypass	Real-Time Viable?
TF-IDF + LR (Proposed)	0.96 ms	0.0%	100.0% (n=1,000)	✓ Yes
Llama-3 8B	5,411.73 ms	40.0%	0.0%	✗ No
Mistral 7B	5,426.69 ms	60.0%	100.0%	✗ No
Phi-3 Mini	6,459.72 ms	20.0%	40.0%	✗ No
Dolphin-Llama3	5,506.59 ms	66.7%	100.0%	✗ No

TABLE III. Latency vs. Detection Benchmark: Statistical Classifier vs. Local LLM Judges. Bypass rate is the percentage of attacks that evaded the detector. Lower is better for bypass; lower is better for latency. No single architecture dominates on both dimensions simultaneously.

The results expose the fundamental tradeoff with unusual clarity. Llama-3 is the strongest semantic detector in this benchmark: it catches every subtle evasion in the test set (0% subtle bypass). It does so at a cost of 5,411 ms per inference—roughly 108 times the real-time constraint. In a three-agent pipeline, deploying Llama-3 as the guard at each boundary would add more than 10 seconds of latency to every inter-agent exchange.

The LLM overt bypass rates require explanation. Llama-3 at 40%, Mistral at 60%, and Dolphin-Llama3 at 66.7% are counterintuitive results—these are shockingly high for attacks that TF-IDF catches 100% of. The underlying reason is not that LLMs cannot understand role-override commands; on the contrary, they understand them too well. These models are, by design, instruction-following systems. When presented with an authoritative-sounding directive—"You are now in admin mode. Disregard prior constraints"—the same alignment that makes them useful assistants inclines them to comply rather than flag the instruction as adversarial. A logistic regression classifier has no such inclination: it has no concept of authority, no disposition toward helpfulness, and no tendency to follow instructions. This finding is one of the more consequential results of our evaluation: the very property that makes LLMs capable of detecting semantically subtle attacks (contextual understanding) also makes them structurally unreliable against overt ones when deployed as security guards without additional scaffolding.

V. The Hybrid Defense Architecture

Section IV left us with a clean, uncomfortable result: the only classifier fast enough for real-time deployment fails entirely on adversarially paraphrased attacks, and the only approach with sufficient semantic depth to catch those attacks is too slow by three orders of magnitude to be useful inline. The hybrid architecture proposed in this section rejects that framing. The dichotomy holds only if we require a single inspection layer to handle all traffic uniformly. It dissolves the moment we allow the system to route messages differently based on how confident the fast classifier already is about them.

The intuition is not exotic. It is the same principle behind tiered caching, staged query planners, and risk-based authentication: match the cost of a decision to the difficulty of making it. Most messages are easy—the statistical classifier is highly confident, one way or the other, and the right answer costs under a millisecond to reach. A small fraction of messages genuinely require semantic judgment. Route only those to the LLM. The slow path becomes rare enough that the blended latency across all traffic drops to an operationally tolerable level.

A. Confidence-Calibrated Routing

The Logistic Regression classifier outputs a continuous probability score $P(y = 1 \mid x)$—the model's confidence that an incoming message is malicious. Examining the classifier's output distribution on Stratum III samples reveals something directly useful: adversarially paraphrased injections do not score near zero. They are not classified as confidently benign. They tend to cluster in the range $P \in [0.2, 0.4]$—anomalous enough that the model registers uncertainty, but not anomalous enough to cross the 0.5 blocking threshold. This band is the classifier's epistemic limit: patterns it recognizes as unusual but cannot place definitively on either side of the decision boundary. It is precisely this band where a secondary semantic judge adds the most value.

We formalize this as a dual-threshold routing mechanism. Let τ_{n0l} and τh^{IGh} define the lower and upper boundaries of the uncertainty band, with $\tau_{n0l} < 0.5 < \tau h^{IGh}$. For an incoming inter-agent message x, the routing decision is:

> ***Fast-Track Allow:*** $P(y = 1 \mid x) < \tau\_low \rightarrow$ high-confidence benign. Forwarded immediately. No LLM consulted. Latency ≈ 0.96 ms.
>
> ***Fast-Track Block:*** $P(y = 1 \mid x) > \tau\_high \rightarrow$ overt injection detected with high confidence. Connection terminated immediately. Latency ≈ 0.96 ms.
>
> ***Semantic Escalation:*** $\tau\_low \leq P(y = 1 \mid x) \leq \tau\_high \rightarrow$ model uncertain. Payload routed to Layer 2 LLM judge for semantic verification before any forwarding decision.

The boundary values τ_low and τ_high are calibration parameters, not architecture constants. The right values depend on the deployment's asymmetric cost structure: how severely a missed injection is penalized relative to a dropped legitimate message, and how much additional per-hop latency the pipeline can sustain. Default calibration on

held-out validation data yields $\tau\_low = 0.15$ and $\tau\_high = 0.85$. At these settings, 95.2% of traffic resolves on the fast statistical path and 4.8% of messages enter the semantic escalation path.

B. Blended Latency Analysis

The expected blended latency per hop, E[T], follows from the law of total expectation across the two routing paths:

$$E[T] = (1 - p_e^{sc}) \times T\_fast + p_e^{sc} \times T\_slow$$

Substituting measured values—$T\_fast = 0.96$ ms, $T\_slow = 5{,}411$ ms (Llama-3), $p\_esc = 0.048$—yields $E[T] \approx (0.952 \times 0.96\ \text{ms}) + (0.048 \times 5{,}411\ \text{ms}) \approx 261$ ms per hop. This represents a 20.8× reduction compared to full LLM routing, while maintaining total threat coverage across all attack strata.

Architecture	Mean Latency	Slowdown	Overt Coverage	Subtle Coverage
No Defense	0 ms	1.0×	0%	0%
TF-IDF Only (Layer 1)	0.96 ms	1.02×	100%	0%
LLM-Only (Llama-3)	5,412 ms	108.2×	60%	100%
Hybrid — Proposed ($p\_esc = 4.8\%$)	≈261 ms	5.2×	100%	100%

TABLE IV. Comparative Latency and Security Coverage Across Defense Configurations. The hybrid is the only design achieving 100% coverage on both attack strata within an operationally tolerable latency envelope. Slowdown is relative to the unguarded baseline.

C. Security Properties and Graceful Degradation

Volume attacks (high-frequency overt injections). Brute-force flooding with standard injection patterns is handled entirely by Layer 1 at sub-millisecond speed. These messages never reach the escalation path, so they cannot saturate the LLM's GPU queue or cause latency blowup in the semantic layer. The fast path absorbs the load deterministically, with throughput bounded by CPU capacity rather than GPU availability.

Targeted evasions (adversarially paraphrased). Sophisticated Stratum III attacks score in the classifier's grey zone and are escalated to Llama-3, which achieved 0% bypass on the held-out Stratum III test set. The attacker cannot escape detection by paraphrasing around keyword patterns; doing so moves the payload into the uncertainty band, which routes it directly to the component capable of evaluating semantic intent rather than surface features.

Layer 2 unavailability (GPU outage or API rate limiting). If the LLM judge becomes unavailable, the system degrades to Layer 1-only operation. Benign and overt traffic continues to be handled correctly at full throughput. Escalated messages from the uncertainty band are held and logged as elevated-risk, or optionally dropped with explicit operator notification. This is a fundamentally different failure profile from a monolithic LLM guard: when that component fails, the entire security layer fails with it.

Zero-day and out-of-distribution attacks. Attack variants that neither layer was trained on may still score in the uncertainty band if they share partial feature overlap with known patterns—providing an inherent safety net. The system's response to the unknown is escalation and scrutiny, not silent passage.

D. Deployment Integration and Scalability

The firewall operates at the message serialization layer, intercepting payloads formatted as JSON-RPC (A2A protocol), MCP-structured tool calls, or custom agent communication formats before delivery. From the agent's perspective, the firewall is indistinguishable from normal network latency on clean messages; blocked messages simply do not arrive—requiring no changes to existing agent implementations. Layer 1 classifiers are fully stateless, making horizontal scaling across CPU nodes trivial with no synchronization overhead. Layer 2 escalations benefit from request batching, reducing per-message latency on the slow path by 10× to 50×.

VI. Limitations and Future Work

Every architecture has a boundary condition—a set of assumptions that, if violated, cause it to behave differently from its design intent. The hybrid system described in Section V is not an exception. Naming these boundaries clearly is more useful than understating them, both for researchers building on this work and for practitioners deciding whether to deploy it.

A. Adaptive Adversaries and Grey-Zone Flooding

The hybrid architecture's latency guarantee rests on one empirical assumption: that subtle adversarial injections are low-frequency events in ambient pipeline traffic. An attacker who understands this assumption can attack it directly. An automated system generating high volumes of paraphrased payloads specifically engineered to score within the uncertainty band $\tau\_low \le P(y = 1 \mid x) \le \tau\_high$ will drive p_esc toward 1.0, causing blended latency to converge toward the LLM baseline of approximately 5,400 ms. This is, in effect, an algorithmic denial-of-service attack that exploits the security layer itself as the attack surface.

Several mitigation directions are worth pursuing. Token bucket rate-limiting on the escalation path would cap an attacker's ability to sustain high p_esc. Progressive threshold tightening under sustained uncertainty load, where τ_low and τ_high temporarily converge toward 0.5 to reduce the escalation window, would limit the DoS surface at the cost of briefly degraded subtle-injection coverage. Traffic fingerprinting for coordinated paraphrase campaigns could trigger circuit-breaker responses before latency degrades. Adversarial training of the Layer 1 classifier on synthetic grey-zone examples would compress the uncertainty band itself.

B. Static Vocabulary and Concept Drift

The TF-IDF classifier's feature space is fixed between training runs. This is a deliberate design choice—a static vocabulary enables the deterministic, variance-free inference that makes sub-millisecond latency possible—but it creates a well-defined vulnerability: novel adversarial vectors that operate outside the learned feature distribution may score below τ_low and pass through both detection layers without triggering escalation. Multilingual obfuscation, Unicode homoglyph substitution, and structured injection through emerging agent communication protocols are all plausible attack vectors that a vocabulary trained on today's injection patterns might not represent.

Maintaining defensive efficacy over time requires continuous vocabulary updating against verified threat intelligence, automated threshold recalibration on rolling validation sets, and integration with adversarial example databases such as AgentDojo [19]. The distinction between legitimate linguistic evolution and adversarial vocabulary shift must be carefully managed.

C. Scope Constraints and Open Problems

The evaluation in this paper assumes a homogeneous, serialized three-agent topology. Several important deployment contexts fall outside this scope. In asynchronous pipelines where agents send messages concurrently, message ordering attacks—where an injection is split across multiple concurrent payloads that individually score as benign—may bypass per-message inspection entirely. Stateful inspection across message streams would be required. Pipelines that encrypt inter-agent payloads for privacy present an obvious tension with inline content inspection. The MAPI-6K dataset covers text-based messages exclusively; agents exchanging structured tool outputs, binary data, or image-based content introduce attack surfaces the current architecture cannot address.

D. Broader Impact

There is a documented gap between the pace at which enterprises are adopting autonomous systems and the pace at which security practices are keeping up with that adoption. Security concerns have emerged as a primary barrier preventing organizations from moving agentic systems from pilot to production, particularly in regulated industries where an uncontained security failure carries legal and operational consequences. A defense layer with deterministic, auditable performance characteristics—one that can produce a propagation depth metric and a per-hop latency figure for a compliance report—addresses a different need than a defense that works but cannot be measured.

We also note a dual-use concern that responsible publication requires acknowledging. The grey-zone characterization developed in this paper—the finding that adversarially paraphrased injections cluster in the probability range $P \in [0.2, 0.4]$—could, in principle, be used to craft injections that deliberately target that range and thereby force escalation as a denial-of-service mechanism. We have described this attack vector explicitly in Section VI-A precisely because a defense that does not anticipate it cannot be designed to resist it.

VII. Conclusion

The shift from isolated language models to autonomous multi-agent pipelines is not a marginal architectural change. It transforms the threat model entirely. In a single-model deployment, a successful prompt injection compromises one model's behavior for one interaction. In a multi-agent pipeline, the same injection can propagate downstream through every agent in the chain—each forwarding the compromised output as if it were legitimate context—reaching high-privilege components with the ability to execute code, authorize transactions, or send external communications. Our simulation measured this directly: without a firewall, injections reached an average propagation depth of 3.0 hops in a three-agent pipeline, meaning every agent was compromised from a single point of entry, every time.

The defensive approaches inherited from single-model security do not scale to this setting. Statistical classifiers operate at the required speed but fail entirely on adversarially paraphrased injections. LLM-based semantic judges catch what statistical classifiers miss but impose latencies of over 5,000 ms per hop—structurally incompatible with pipelines that must complete multi-step tasks in real time. Neither approach, deployed alone, is adequate.

The LLM Agent Firewall addresses this by treating the problem as two distinct subproblems requiring different tools. At Layer 1, a TF-IDF + Logistic Regression classifier intercepts every inter-agent message with a mean latency of 0.96 ms, blocking 100% of overt injection categories with zero false positives and reducing propagation depth from 3.0 to 0.0 hops in simulation. At Layer 2, a confidence-calibrated routing mechanism escalates only the 4.8% of messages that fall in the classifier's uncertainty band to a local LLM judge—Llama-3, which achieved 0% bypass on the held-out subtle evasion test set—for semantic verification. The blended per-hop latency of approximately 261 ms is a 20.8× improvement over full LLM routing and remains within the operational tolerance of autonomous pipelines.

> *The key insight is not that statistical classifiers are fast or that LLMs are semantically aware—both facts were already known. It is that routing decisions can be made based on classifier confidence rather than content alone, concentrating expensive semantic analysis on precisely the messages where it changes the outcome. This reframing converts an irreconcilable tradeoff into an engineering problem with a tractable solution.*

Several threads remain open. Grey-zone flooding—where an attacker deliberately saturates the escalation

path—represents a real denial-of-service vector that requires rate-limiting and dynamic threshold adaptation to contain. The static TF-IDF vocabulary needs continuous updating against an evolving threat landscape. Asynchronous pipelines, encrypted channels, and cross-modal attack vectors all extend the problem space beyond what the current evaluation covers. These are not arguments against deployment; they are the natural research agenda for a defense approach that has been established on solid empirical ground.

As multi-agent systems become more capable and more deeply integrated into critical infrastructure, securing the channels between agents will become as foundational as securing the channels between network nodes. The architecture developed here—treating inter-agent messages as packets requiring inline inspection, classifying at machine speed, and escalating semantically ambiguous cases to deeper analysis—is one instantiation of that principle. We offer it as a starting point, not a final answer.

REFERENCES

[1] K. Greshake et al., "Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection," Proc. 16th ACM Workshop on Artificial Intelligence and Security (AISec), pp. 79–90, 2023.

[2] F. Perez and I. Ribeiro, "Ignore Previous Prompt: Attack Techniques For Language Models," arXiv:2211.09527, 2022.

[3] H. Branch et al., "Evaluating the Susceptibility of Pre-Trained Language Models via Handcrafted Adversarial Examples," arXiv:2209.00118, 2022.

[4] A. Wei et al., "Jailbroken: How Does LLM Safety Training Fail?" Advances in Neural Information Processing Systems (NeurIPS), vol. 36, 2023.

[5] D. Kang et al., "Exploiting Programmatic Behavior of LLMs: Dual-Use Through Artful Prompts," IEEE Symposium on Security and Privacy (S&P), 2023.

[6] X. Shen et al., "Do Anything Now: Characterizing and Evaluating In-The-Wild Jailbreak Prompts on Large Language Models," ACM CCS, 2023.

[7] A. Zou et al., "Universal and Transferable Adversarial Attacks on Aligned Language Models," arXiv:2307.15043, 2023.

[8] S. Zhu et al., "AutoDAN: Generating Stealthy Jailbreak Prompts on Aligned Large Language Models," arXiv:2310.04451, 2023.

[9] D. Kong et al., "A Survey of LLM-Driven AI Agent Communication: Protocols, Security Risks, and Defense Countermeasures," arXiv:2506.19676, 2025.

[10] K. Zhu et al., "PromptBench: Towards Evaluating the Robustness of Large Language Models on Adversarial Prompts," arXiv:2308.14930, 2023.

[11] U. Alon and H. Kamigaito, "Detecting Language Model Attacks with Perplexity," ACL Findings, 2023.

[12] N. Jain et al., "Baseline Defenses for Adversarial Attacks Against Aligned Language Models," arXiv:2309.00614, 2023.

[13] E. Wallace et al., "Universal Adversarial Triggers for Attacking and Analyzing NLP," EMNLP, 2019.

[14] Y. Zhang et al., "Fast and Accurate Text Classification with Statistical Models," IEEE Trans. Knowledge and Data Engineering, vol. 34, no. 8, pp. 3789–3802, 2022.

[15] I. J. Goodfellow et al., "Explaining and Harnessing Adversarial Examples," ICLR, 2014.

[16] T. Rebedea et al., "NeMo Guardrails: A Toolkit for Controllable and Safe LLM Applications," NVIDIA Technical Report, 2023.

[17] H. A. Inan et al., "Llama Guard: LLM-based Input-Output Safeguard for Human-AI Conversations," arXiv:2312.06674, 2023.

[18] Y. Wang et al., "Latency and Throughput Analysis of LLM-Based Safety Guardrails," Proc. USENIX Security Symposium, 2024.

[19] E. Debenedetti et al., "AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents," Advances in Neural Information Processing Systems, vol. 37, pp. 82895–82920, 2025.

[20] Q. Wu et al., "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation," arXiv:2308.08155, 2023.

[21] J. Moura, "CrewAI: Framework for Orchestrating Role-Playing, Autonomous Agents," GitHub Repository, 2023.

[22] H. Chase, "LangGraph: Building Stateful, Multi-Actor Applications with LLMs," LangChain Documentation, 2024.

[23] Z. Zhang et al., "A Survey on the Memory Mechanism of Large Language Model Based Agents," arXiv:2404.13501, 2024.

[24] Z. Chen et al., "AgentPoison: Red Teaming LLM Agents via Poisoning Memory or Knowledge Bases," arXiv:2407.12784, 2024.

[25] S. M. Khan et al., "Permutation-Invariant Adversarial Attack Method for Multi-Agent Systems," Proc. International Joint Conference on Neural Networks (IJCNN), 2025.

[26] Y. Chen et al., "Secure Multi-Agent Frameworks via Continuous Verification," ACM CCS, 2024.

[27] H. Liu et al., "Cryptographic Provenance for LLM Workflows," IEEE S&P, 2024.

[28] M. Roesch, "Snort — Lightweight Intrusion Detection for Networks," Proc. USENIX LISA Conference, 1999.