

AURA: A Lightweight, CPU-Only Hybrid Defense for Deployable Prompt Injection Detection

Mr. Santosh S. Kare¹, Harshad Ghutukade², Ashish Kamble³, Manthan Mane⁴, and Om Magar⁵

Department of Information Technology,

Vidya Pratishthan's Kamalnayan Bajaj Institute of Engineering and Technology, Baramati, Pune.

¹santosh.kare@vpkbiat.org ²harshadghutukade9016@gmail.com, ³ashishkamble0404@gmail.com,

⁴manthanmane07@gmail.com, ⁵ommagar414@gmail.com

Abstract. Prompt injection represents one of the main security vulnerabilities in LLM-powered applications, including autonomous agents operating in complex environments. State-of-the-art approaches for prompt injection detection rely on expensive fine-tuning on large LLMs, causing long inference times, GPU dependence, and increased False Positive Rate (FPR). Our proposed approach, named AURA (Adaptive Universal Retrieval Augmentation), is aimed at developing a lightweight and CPU-only middleware model-agnostic prompt injection detector. This method consists of three main components: (1) precision-based regex filters for detecting jailbreaks and exfiltration attempts, (2) anomaly detectors based on structural information such as entropy and length statistics and (3) semantic intent clustering using zero-shot methods with small embedding size (BAAI/bge-small-en-v1.5). During extensive evaluations on our dataset of 60 carefully crafted adaptable prompt injections and 125 benign prompts (including 15 hard-to-distinguish Benign Utility queries) we achieved True Positive Rate (TPR) of 90.0%, FPR of 0.00%

, Accuracy of 96.8% and Benign Utility Score of 100.0% at average latency 12.1 ms (P99 22.0 ms) in CPU-only settings. We were also able to detect 60.0% of indirect prompt injection attacks provided by BIPIA, at the cost of lower absolute TPR rates in favor of deploying lightweight defenses and ensuring minimal FPR rates. We conclude that even simple semantic and structural filters provide an efficient solution as an external layer to protect LLMs against prompt injections.

Keywords: Prompt Injection, Large Language Models, Adversarial Attacks, Security, Anomaly Detection, Semantic Similarity, CPU Inference.

1 Introduction

The rapid deployment of Large Language Models (LLMs) into consumer and enterprise environments has resulted in the creation of an entirely new vulnerability surface known as prompt injection. The prompt injections consist of malicious actors designing certain prompts that change the context of instructions for the model, thereby permitting the attackers to steal data, run any program, and take control of the system by feeding the model with toxic input from outside. Studies have found that prompt injections can compromise both real-life systems and benchmarking tests, showing that LLMs are susceptible to such attacks [1][2][3][4].

Existing defenses often employ secondary LLMs, complex multi-agent pipelines, or model-internal detectors to evaluate input safety [2][5][6]. While these approaches can significantly reduce attack success rates (ASR), they introduce:

- Latency issues, higher computational costs (need for GPU).
- Dependence on the details of the inner workings of the model.
- False-positive rate (FPR), which negatively affects the user experience [5][6].

In the cases of edge computing, API gateway, and high-throughput systems, however, this overhead is not acceptable. Some practical use cases will need:

- CPU-only operation,
- model-agnostic external guards,
- strict FPR targets ($\approx 0\text{--}1\%$) to preserve utility, and
- per-request latency in the low-millisecond range.

In this paper, we present AURA, a three-layer hybrid detector for prompt injections. AURA is developed as an external middleware that performs prepro-cessing on the inputs and classifies them prior to forwarding to the generative model with the following objectives in mind:

1. Deployability: CPU-only, minimal memory footprint, no access to model internals.
2. Low FPR / High Utility: Target $0\text{--}1\%$ FPR with high benign pass-through.
3. Reasonable Coverage: Detect a broad range of curated and adaptive prompt injections, including indirect attacks, with acceptable TPR.
4. Low Latency: Maintain sub-15 ms average latency under realistic workload.

AURA employs highly-precise lexicons, anomaly detection based on structural features, and lightweight semantic similarity checking inspired by the recent evidence that feature ensembles and small embedding classifiers have great potential [5][6][2].

Our contributions include:

- Design of AURA, a CPU-only, tri-layer hybrid detector for prompt injection.
- Quantitative evaluation on curated, benchmark-derived, and adaptive attacks, as well as benign and complex benign prompts.
- Analysis of indirect attack coverage, including BIPIA’s indirect prompt injection scenarios [2].
- Discussion of limitations and trade-offs relative to heavier, model-internal or multi-agent defenses [2][6][7][8].

2 Related Work

2.1 Indirect Prompt Injection and Benchmarks

Indirect prompt injection attacks leverage the fuzziness between data and instructions in LLM-powered applications [1][3][9]. Benchmarks such as BIPIA provide a formal approach to modeling such attacks, illustrating that all LLM models are equally susceptible if external data is considered as instruction-loaded input [2]. BIPIA tests both black-box and white-box defenses by designing strategies such as boundary awareness and reminders, which indicate that white-box defenses can effectively lower ASR scores close to zero without degrading outputs [2].

For agents that utilize tools and external sources, InjecAgent examines 1,054 test scenarios across 17 user tools and 62 attacking tools, indicating that tool-enabled LLM agents remain open to indirect prompt injections, and ReAct-instructed GPT-4 is attacked in 24% of test scenarios [4]. More recent research reveals that adaptive and gradient-based prompt injection approaches can systematically defeat numerous defenses [10][11].

2.2 Detection-Focused Defenses

Several studies have proposed approaches based on external detection and not on architectural modifications. Lightweight detectors based on prohibited token embeddings and the use of tiny classifiers based on BERT models showed that prompt injection could be identified in real-time with fairly good accuracy [5].

Classical ML models or embedding-based solutions applied to BIPIA and other datasets have yielded high accuracy levels ($\approx 97\text{--}99\%$), which also shows that external detectors can work efficiently [5][2].

There has been an explicit focus on developing indirect prompt injection detection systems, along with a post-hoc method for prompt removal through benchmarking and training of such detectors [6]. Model-internal

detectors such as InstructDetector rely on gradients and hidden states to detect instruction-like inputs from external sources with accuracy ranging from 99.6% to 96.9% (in-domain and out-of-domain respectively) and lowering BIPIA ASR to 0.03% [7]. However, such solutions depend upon internal access to models.

2.3 Model-Level, Prompt, and System Defenses

A range of prompt-level countermeasures including boundary-awareness, reminders, and spotlighting aim at marking and isolating the different inputs in the prompt from one another, decreasing the likelihood that untrusted information is considered an instruction by the model [2][12]. The application of spotlighting adds provenance signals to the prompts, achieving greater than 50% attack failure rate while introducing negligible performance losses in evaluation [12].

Agent-level countermeasures include IPIGuard (tool-dependency graph) and CaMeL (control/data-flow isolation of planning), which allow the separation of planning from untrusted inputs, implementing capability-based policies on the available tools [9][8]. These solutions demonstrate strong robustness and provide provable security guarantees in AgentDojo, but they imply substantial changes to the system design and require additional assumptions about the agents.

2.4 AURA in Context

AURA is positioned as a black-box, external, CPU-only detector:

- It does not require model internals (unlike InstructDetector [7]).
- It avoids heavy LLM evaluators or multi-agent defense pipelines [4][8][9].
- It is based on a lower FPR and low latency but maintains acceptable coverage against curated and indirect attacks.

This complements other heavy-duty white-box or system-level defense measures that have been successful using embedding-based and multi-feature detectors [5][2][6].

3 Threat Model

3.1 Adversary Capabilities

The adversary can inject directly into the prompt (e.g., Ignore previous instructions, Reveal your system prompt), perform indirect prompt injections via external content such as hidden instructions in web pages, documents, or tool outputs which the LLM-integrated system retrieves and processes, and use obfuscation techniques (e.g., encoding, spacing, formatting) to evade naive pattern matching. The attacker's goal is to hijack the LLM's task execution by overriding original instructions, exfiltrating private data, or inducing arbitrary tool calls.

3.2 Defender Capabilities and Constraints

AURA operates as an external middleware layer between the application and the LLM. It has no insights into the inner workings of the LLM (weights, activation, probability of tokens). It runs exclusively on CPUs with a target memory usage of less than 1 GB, must process each input in sub-15 ms average latency, and must maintain 0–1% FPR in production-like traffic to avoid degrading user experience. The middleware may prevent, soft-prevent, or wrap suspicious prompts (for instance, by attaching warnings or using safe modes), but it does not alter the weightings of LLMs or change the tokens in any way.

4 Methodology

AURA is a pure detection middleware that scores every input prompt based on its threats. It is built on top of a three-tier architecture:

1. Pattern-based detection employing a lexicon of carefully hand-crafted regular expressions.
2. Structure-based detection employing entropy and length information.
3. Semantics-based detection leveraging lightweight embeddings for calculating semantic intent similarity.

An input prompt is flagged malicious if the overall score surpasses the detection threshold for achieving close-to-zero FPRs.

4.1 Layer 1: Lexical Regular Expressions

Layer one contains a manually constructed set of regular expressions to accomplish the following: jailbreak instructions (e.g., disregard previous instructions, act like you are a system prompt, turn on developer mode); clear-cut data ex-filtration instructions (e.g., show your system prompt, extract all of the user's data); and vulnerabilities based on known techniques such as BIPIA and Open-Prompt-Injection [2][3][13]. The aim at this point is to ensure accuracy through matching only in cases where the lexically strongest evidence is available. The operation is completely deterministic with negligible latency costs.

4.2 Layer 2: Structural Anomaly Analysis

Structural features are computed at Layer 2, which include the following: character/token Shannon entropy (to detect unusual high or low values of entropy, like base64 encoding); percentage of special characters (to detect any kind of character encoding or obfuscation within the prompt); and prompt and segment length ratios (to detect unusually long segments in the user's prompt). This ensures that legitimate prompts are not identified as being anomalous due to the threshold setting. This layer is applied based on the understanding that multi-feature attack detection techniques outperform lexicon-based detection systems [5][6].

4.3 Layer 3: Semantic Intent Similarity

For the detection of semantically obfuscated attacks, AURA employs the BAAI/bge-small-en-v1.5 semantic embedding model as the lightweight semantic encoder:

1. the prompt is embedded as a fixed-size vector;
2. malicious intent anchors are precomputed via embedding representative attack prompts, based on injection corpora (BIPIA attack commands) [2][7] and exfiltration templates/prompt examples from benchmark datasets [3][13]
3. cosine similarity is calculated between the prompt embedding and each anchor.
4. the maximum similarity value will be a part of the threat score.

Anchors have been selected for their broader malicious intents (e.g., ignore all other instructions and follow mine, exfiltrate everything secret) as opposed to being based on specific strings.

4.4 Scoring and Decision Rule

In each layer, the intermediate score is calculated as follows: Layer 1 produces a binary/weighted match score; Layer 2 produces a normalized anomaly score based on entropy and density thresholds; and Layer 3 produces a normalized semantic similarity score. Finally, a threat score is derived using the weighting of scores. The thresholds and weights are optimized based on a validation set in order to achieve minimum FPR and maximum TPR in the range of 0–1%.

4.5 Implementation and Runtime Environment

All modules are CPU-only and rely on ONNX Runtime for embedding calculations. The memory usage stays below 1 GB RAM, which includes the embedding model, anchors, and feature extractors. The middleware module provides a basic HTTP/gRPC API interface to upstream applications.

5 Experimental Setup

5.1 Datasets

AURA is evaluated on a dataset that consists of the following subsets:

- **Curated direct prompt injections (50 samples).** Including Deepset-style injection templates and jailbreaks aimed at instruction override and exfiltration.
- **BIPIA-derived indirect attacks (75 samples).** Indirect injection instructions embedded in other content including 15 types of attacks with task-irrelevant, relevant, and target-specific instructions [2][7].
- **Custom adaptive attacks (10 samples).** Including attacks based on encoding (base64) of injection content; indirection and paraphrasing to bypass regex detection.
- **Benign prompts (110 samples).** Normal user queries related to multiple tasks such as summarization, advice, QA, etc., without any injection.
- **Benign Utility set (15 samples).** Very complex yet benign prompts that can be useful to a human user in some scenarios (complex multi-step instructions).

We use 60 attack prompts and 125 benign prompts for computing the main metric.

5.2 Hardware and Execution

All experiments are conducted on a local CPU-only machine (no GPU). Embedding inference uses ONNX Runtime, optimized for CPU. Latency is measured end-to-end from middleware input to detection decision.

5.3 Metrics

The following metrics are used for evaluation:

- **True Positive Rate (TPR):** percentage of attack instances successfully detected.
- **False Positive Rate (FPR):** percentage of benign prompts wrongly classified as attack instances.
- **Accuracy:** percentage of correctly classified prompts.
- **Benign Utility Score:** percentage of Benign Utility samples that are not blocked.
- **Latency:** average latency and 99th percentile latency for each request.

The FPR objective of 0–1% is enforced, in line with current detectors used in production environments and existing FPR objectives [5][6].

6 Results

6.1 Overall Detection Efficacy and Utility

On a combined dataset of curated and adaptive samples (60 attacks and 125 benign prompts), AURA delivers the following results: True Positive Rate (TPR) of 90.0%, False Positive Rate (FPR) of 0.00%, Benign Utility Score of 100.0%, and Accuracy of 96.8%. This means correctly stopping 54 of 60 attacks, incorrectly blocking zero of 125 harmless prompts, and passing all 15 difficult Benign Utility prompts. These results are similar to those achieved by machine learning classifiers trained for indirect prompt injection in low-FPR settings [5][6][2] but without the need for heavy models.

6.2 Performance on Indirect Attacks (BIPIA)

For the 75 BIPIA indirect prompt injection instances [2]: detected attacks: 45; de-tection rate: 60.0%. BIPIA's own defense mechanisms, such as instruction-state detectors, may lower the success rate of attacks to 0.03–0.12% [2][7]. However, the performance of AURA is impressive at 60.0%, despite being a pure black-box zero-shot system.

Table 1. Summary of AURA performance on attack and benign datasets

Dataset/Subset	Sampl es	Detection Rate/Score	Notes
Curated + adaptive attacks	60	TPR 90.0%	External input only
Benign prompts	125	FPR 0.00%	Includes everyday queries
Benign Utility set	15	Utility 100.0%	Complex, non-malicious
BIPIA indirect attacks	75	Detection 60.0%	Benchmark indirect IPI [2][7]

6.3 Latency

Measured end-to-end per request, the mean latency is approximately 12.1 ms and the P99 latency is 22.0 ms. This measure encompasses structural analysis and embeddings inference running on single-threaded CPU. This is many mag-nitudes faster than common LLM evaluation workflows or remote API-based safety systems [2][4][6], hence making AURA well-suited for high-throughput scenarios.

7 Discussion

7.1 Trade-offs: Deployability vs. Coverage

AURA's design ideology focuses on deployment and low FPR rather than com-pleteness. While white-box defense mechanisms using internal states and gradi-ents can provide ASR of 0.03% to BIPIA attacks [7][2], the detection ability of AURA to detect BIPIA attack samples stands at only 60%, which is compara-tively lower. However, AURA provides 0.00% FPR and 100% utility for benign samples of benchmark datasets, while remaining CPU-only and model-agnostic.

Therefore, AURA is the ideal solution to serve as the middleware security mechanism for any environment that cannot use other methods.

7.2 Complementarity with Existing Defenses

AURA can serve as a first-line filter before more expensive defenses (e.g., instruction-state detectors [7], spotlighting [12], system-level sandboxes [8][9]). It can reduce the load on downstream heavy defenses by filtering obvious or high-risk inputs early, and can coexist with prompt-level defenses (e.g., boundary prompts, signed prompts) [2][14], providing an orthogonal signal based on semantics and struc-ture.

8 Limitations and Future Work

There are still some limitations of the current system:

- **Indirect Attack Coverage.** The AURA system can attain coverage of up to 60% against BIPIA indirect attacks, but this is significantly lower compared to specialized white-box defense mechanisms that can reduce ASR to almost zero [2][7]. Increasing coverage on indirect attacks without raising the FPR remains an open challenge.

- **Adaptive and Optimization-Based Attacks.** Recent studies have shown that both adaptive and optimization-based attacks (such as automatic uni-versal injections) are capable of consistently evading heuristics and embeddings, which could lead to severe overestimation of robustness in models without thorough testing against such attacks [10][11]. AURA has not been evaluated systematically against such adaptive opponents.
- **Detection vs. Removal.** AURA strictly separates the detection problem from the removal task. It does not perform any kind of removal in either segment-level or sentence-level, unlike some other recent work that tries to extract the benign text while removing the injected instructions [6].
- **Dataset Size and Diversity.** The current evaluation experiments use a relatively small number of curated, benchmark-derived, and adaptive attacks. Larger evaluations using publicly available benchmarks (Open-Prompt Injection [3][13] or InjecAgent [4]) and realistic traffic patterns are worth exploring in future work.

9 Conclusion

This paper introduced AURA, a lightweight, CPU-only, tri-layer hybrid approach to detect prompt injections in LLM-powered systems. Through leveraging highly accurate regex filters, analyzing structural anomalies, and comparing semantics via small models, AURA reaches 90.0% TPR, 0.00% FPR, 96.8% accuracy, and 100.0% Benign Utility on an aggregate curated and adaptive dataset with approximately 12.1 ms mean latency in pure-CPU runtime. For BIPIA-based attacks, AURA detects 60.0% of the attacks without having access to model internals.

Instead of substituting heavyweight white-box or system-level defenses, AURA provides practical model-agnostic middleware shielding for scenarios where such defenses would be impractical. Our future research plans include improvements in handling indirect and adaptive attacks, studying integration with attack removal approaches, and benchmarking on publicly available datasets.

References

1. Greshake, K., 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 AI and Security (2023)
2. Yi, J., et al.: Benchmarking and Defending against Indirect Prompt Injection Attacks on Large Language Models. In: Proc. 31st ACM SIGKDD (2023)
3. Liu, Y., et al.: Prompt Injection Attack against LLM-Integrated Applications. arXiv:2306.05499 (2023)
4. Zhan, Q., et al.: InjecAgent: Benchmarking Indirect Prompt Injections in Tool-Integrated Large Language Model Agents. arXiv:2403.02691 (2024)
5. Lan, Q., Kaul, A., Jones, S.: Prompt Injection Detection in LLM Integrated Applications. International Journal of Network Dynamics and Intelligence (2025)
6. Chen, Y., et al.: Can Indirect Prompt Injection Attacks Be Detected and Removed? (2025)
7. Wen, T., et al.: Defending against Indirect Prompt Injection by Instruction Detection. arXiv:2505.06311 (2025)
8. Hines, K., et al.: Defending Against Indirect Prompt Injection Attacks With Spot-lighting. arXiv:2403.14720 (2024)
9. An, H., et al.: IPIGuard: A Novel Tool Dependency Graph-Based Defense Against Indirect Prompt Injection in LLM Agents (2025)
10. Debenedetti, E., et al.: Defeating Prompt Injections by Design. arXiv:2503.18813 (2025)
11. Zhan, Q., et al.: Adaptive Attacks Break Defenses Against Indirect Prompt Injection Attacks on LLM Agents (2025)
12. Liu, X., et al.: Automatic and Universal Prompt Injection Attacks against Large Language Models.

arXiv:2403.04957 (2024)

13. Liu, Y., et al.: Formalizing and Benchmarking Prompt Injection Attacks and De-fenses (2023)
14. Greshake, K., et al.: More than You've Asked For: A Comprehensive Analysis of Novel Prompt Injection Threats to Application-Integrated Large Language Models. arXiv:2302.12173 (2023)