

Real-Time Prompt Injection and Guardrails for Large Language Model Applications: A Review of Attacks, Defense Mechanisms, and Research Challenges

Durgude Siddhi Murlidhar¹, Prof. Abhishek Bangar P.², Prof. Karad Vandana A.³

¹²³ Department of Computer Engineering / Jaihind College Of Engineering / SPPU, Pune, India

Abstract- Large language models (LLMs) are now embedded in applications that read emails, documents, web pages and images and that can call external tools, which makes prompt injection one of the most important application-level security threats. Static rules and a warning in the system prompt cannot guarantee protection. This paper presents a focused narrative review of five recent primary studies (2024–2026) on prompt-injection detection, guardrails and security evaluation, supported by background works cited within them. The literature is organised into seven themes: the threat landscape; the attack taxonomy; reasoning-based detection; mutation-based universal detection; scanner-based detection; empirical attack simulation; and quantitative security assurance with guardrail requirements. The reviewed studies report that a chain-of-thought (CoT) detector reached 93.20 % accuracy and 90.13 % F1 with a 14B model on 2,176 test prompts; mutation-based detection reached 86.14 % (text) and 82.90 % (image) accuracy on an 11,000-sample dataset; an open-source scanner reached 89 % accuracy on 200 synthetic prompts; plaintext injection succeeded in 20 of 20 trials for three of four techniques on GPT-4o Mini while encoded variants failed; and a CVSS-weighted assurance metric ranked four model deployments. However, the evidence is dominated by small, synthetic or single-model evaluations; detectors are compared on different datasets against different baselines; latency and query cost are rarely controlled; and none of the reviewed experiments evaluates detection together with action-level controls in a tool-calling agent. The paper contributes a thematic synthesis, a comparison of the five studies, a conceptual eight-stage layered framework and a prioritised research agenda.

Keywords— Prompt Injection; Large Language Models; Guardrails; Real-Time Detection; LLM Security; Jailbreaking; Chain-of-Thought; OWASP

I. INTRODUCTION

Large language models (LLMs) are used in education, healthcare, banking, software development and customer service, where they answer questions, summarise documents and generate code. This capability brings a new security problem: prompt injection, in which malicious instructions manipulate a model into ignoring its intended task or following unauthorised instructions. Consider an assistant asked to summarise an email in five sentences. If the email itself contains an instruction to ignore the request and reveal confidential information, and the model obeys, the application behaves in an unsafe way even though the user did nothing wrong.

The risk is not hypothetical. Karlsen et al. [3] describe a MITRE ATLAS case in which prompt injection

against MathGPT, an application that turns natural-language questions into executable Python code, led to arbitrary code execution. Shi et al. [1] report, citing an empirical study of 36 LLM-integrated applications [6], that 31 of them were vulnerable to prompt injection.

Two families of countermeasures are studied. Real-time detection inspects inputs and retrieved content while the application runs, and guardrails are mechanisms that constrain inputs, context, outputs and actions. Neither is complete: detectors produce false positives and false negatives, and delimiters or defensive prompts do not create an absolute security boundary.

This review aims to (i) synthesise detection and guardrail approaches for prompt injection, (ii)

compare their methods, reported performance and limitations, (iii) propose a literature-grounded conceptual framework, and (iv) identify research gaps that should be closed before such systems are relied upon in high-stakes settings.

II. REVIEW METHODOLOGY

The review is built around five primary studies selected by the authors because they directly address prompt-injection detection, jailbreak and hijacking detection, vulnerability scanning, attack simulation or quantitative security assessment of LLM applications: Shi et al. [1], Zhang et al. [2], Karlsen et al. [3], Pokhrel et al. [4] and Kurniawan and Chandra [5]. They were published between 2024 and 2026 in IEEE Access, ACM Transactions on Software Engineering and Methodology, the Journal of Information Security and Applications, and two Procedia Computer Science conference volumes. Background works (for example [6]–[13]) are cited only for statements that the primary studies themselves make about them, and should be read in the original before being relied upon further.

All figures in this paper were taken from the tables, figures and text of the primary studies. Where a primary study contains an internal inconsistency, it is flagged in Section III rather than silently corrected. Because the five studies were chosen directly rather than through a logged database search, this is a focused narrative review, not a systematic one, and it makes no claim of exhaustive coverage.

III. LITERATURE REVIEW

1. Prompt Injection and the Threat Landscape

An LLM application receives text from several sources. Some of it carries authorised instructions and some is data the model must merely process. Prompt injection exploits the fact that the model may not reliably distinguish the two. Formally, Shi et al. [1] model an attacked prompt as $P' = P + \Delta P$ and its output as $Y' = f(P')$, where ΔP is the injected content and Y' generally differs from the expected output Y . Zhang et al. [2] distinguish two attack goals: jailbreaking attacks, which bypass safety alignment so that the model produces harmful

content, and hijacking attacks, which make the system perform an attacker-chosen task, for example returning “Hire him” for a résumé regardless of its content.

The OWASP Top 10 for LLM Applications [11] lists prompt injection as LLM01 in the version used by Karlsen et al. [3] (v1.1), alongside insecure output handling, training data poisoning, model denial of service, supply chain vulnerabilities, sensitive information disclosure, insecure plugin design, excessive agency, overreliance and model theft. Kurniawan and Chandra [5] follow the 2025 numbering, in which prompt injection is LLM01, sensitive information disclosure is LLM02 and unbounded consumption is LLM10. The two numbering schemes differ, so identifiers must be cited together with the list version.

2. Attack Taxonomy

Direct injection. The user supplies the malicious instruction, for example asking the model to ignore previous instructions [8]. Kurniawan and Chandra [5] test four plaintext techniques of this kind (“Ignore and Print”, “Ignore and Say”, “Nevermind”, “Screaming Stop”), and Zhang et al. [2] collect naive injection, fake completion, ignoring-context, escape-character and combined attacks. Karlsen et al. [3] additionally test adversarial-suffix and encoded-command attacks, citing the adversarial-suffix method of Zou et al. [9].

Indirect injection. The instruction is hidden in external content that the application processes, such as an email, web page or document, so the user may be unaware of it [7]. The user’s request is harmless; the content is the attack vector.

Retrieval-augmented generation (RAG) injection. A document placed in a knowledge base can influence the model when it is retrieved. Pokhrel et al. [4] summarise PANDORA, an indirect jailbreak method based on RAG poisoning, as reaching success rates of 34.8 % on GPT-4 and 64.3 % on GPT-3.5.

Multimodal injection. Instructions or adversarial perturbations are embedded in images. Zhang et al. [2] include visual adversarial examples and

typographic attacks against the multimodal model MiniGPT-4 in their dataset.

Agent and plugin injection. When a model can call tools, an injected instruction can trigger real actions. Karlsen et al. [3] list insecure plugin design and excessive agency among the OWASP risks and recount that the MathGPT incident involved code execution and plugin compromise. Prompt injection thus becomes an application-security problem, not only a content-safety problem.

3. Reasoning-Based Detection: Chain-of-Thought Anomaly Analysis

Shi et al. [1] detect injection by examining the intermediate steps of a CoT reasoning chain. Each step is embedded (bge-m3), step-level consistency is the cosine similarity with neighbouring steps, path-level consistency aggregates these values, and a confidence term uses the average token log-probability. These are combined in a loss $L = \lambda_1 L_c + \lambda_2 L_d + \lambda_3 L_r$ with $\lambda_1 = 0.7$, $\lambda_2 = 0.3$, $\lambda_3 = 0$ and a threshold $\delta = 0.37$; an anomalous step can then be traced back through the reasoning tree. Models of the Qwen2.5 family (0.5B to 14B) were fine-tuned on the OpenO1-SFT dataset to produce reasoning chains and tested on the Deepset and Safe-Guard prompt-injection test sets (2,176 prompts: 710 injected, 1,466 benign).

With the 14B model the method reached 93.20 % accuracy, 85.57 % precision, 95.21 % recall and 90.13 % F1, against 86.96 % accuracy and 82.48 % F1 for the model prompted to classify directly, and 92.04 % accuracy and 86.74 % F1 for LLM Guard. Accuracy rose with model size (73.07 % at 0.5B to 93.20 % at 14B). The 14B result exceeds LLM Guard by 1.16 points of accuracy and 3.39 points of F1, but the 7B result (88.24 % accuracy) does not; recall is nearly identical to LLM Guard (95.21 % vs 95.13 %), so the F1 gain comes from precision. The authors report that inference latency rises from about one second for the plain model to several seconds or more than ten seconds with CoT.

Points for readers: the 217-sample validation split used to set the hyperparameters is not stated to be excluded from evaluation, and the confusion

matrices in the paper sum to the full 2,176 samples. The text also refers to a “1.4 billion” parameter model where the tables indicate 14B.

4. Mutation-Based Universal Detection

JailGuard [2] rests on the observation that attack inputs are less robust than benign ones: small perturbations often break an attack and produce markedly different responses, while benign responses stay similar. The detector mutates an untrusted input into N variants ($N = 8$ by default), queries the model, converts responses to vectors, builds a cosine-similarity matrix, converts rows to distributions and computes pairwise KL divergence. An input is flagged if any divergence reaches a threshold θ (0.02 for text on GPT-3.5, 0.025 for images on MiniGPT-4), or if all responses are refusals. It implements 18 mutators (16 random and 2 targeted) and a combination policy: for text, punctuation insertion, targeted insertion and translation with probabilities 0.24, 0.52 and 0.24; for images, random rotation, Gaussian blur and random posterization with 0.34, 0.45 and 0.21.

The evaluation dataset has 11,000 items covering 15 attack types: 4,000 text attacks (2,000 jailbreak and 2,000 hijacking), 400 image attacks, and 6,600 benign items. Against 12 baseline methods, the default policy achieved 86.14 % accuracy (precision 80.58 %, recall 86.10 %) on text, 11.81 points above the best text baseline (SmoothLLM-Swap [12], 74.33 %), and 82.90 % accuracy (precision 74.00 %, recall 88.25 %) on images, 12.20 points above the best image baseline (ECSO, 70.70 %). Two baselines that rely on separating or reinforcing instructions reached only 68.03 % (data prompt isolation) and 70.25 % (prompt learning) accuracy on text with recalls of 30.73 % and 37.60 %, which supports the view that instruction– data separation alone is not sufficient.

The authors also report that accuracy peaks for N between 6 and 14 while recall keeps rising, that targeted mutators lose up to 6 points under an adaptive attack aimed at their content-selection step (200 text prompts), and that heuristic, open-ended benign prompts are a main source of false positives. The method needs extra queries to the model and

uses hyperparameters tuned per model, and the authors recommend re-tuning for new deployments. Note that the paper's RQ1 summary gives 82.09 % for images in one place; Table 4 and the abstract give 82.90 %, which is used here.

5. Scanner-Based Detection

Pokhrel et al. [4] evaluate Vigil [13], an open-source scanner available as a Python library and REST API, whose scan ensemble combines vector-database similarity search, YARA rules and a machine-learning classifier, followed by threat-score aggregation and JSON reporting. The test prompts were generated with GPT-3.5-Turbo in three batches (100, 50 and 50 prompts, half malicious and half benign). Reported accuracy was 89 % on the first batch (44/50 malicious and 45/50 benign correct), 90 % on the second (23/25 and 22/25) and 88 % on the third (21/25 and 23/25), giving 178 of 200 (89 %) overall.

The study is useful as a deployment example, but its evidence is limited: the data are small and synthetic, no other detector is compared, and several sentences in the text disagree with the tables (for example the discussion of false positives and the size of the accuracy change in the third batch, which is -4 % in Table 4 against the first batch and -8 % against the second). Several of the malicious samples listed in the paper's dataset table are requests for deceptive content rather than injection payloads, so the results should not be read as a benchmark of injection detection.

6. Empirical Attack Simulation

Kurniawan and Chandra [5] simulate attacks on GPT-2, GPT-3.5 and GPT-4o Mini. For prompt injection each technique was run 20 times and counted as successful when the output matched the target at above 90 % similarity. On GPT-3.5 the successes were 18, 13, 7 and 9 for the four plaintext techniques; on GPT4o Mini 20, 19, 20 and 20; on GPT-2 none. All Base64, Hex and Rot-13 variants failed on every model. The authors read this as larger models being more susceptible, but GPT-2 may simply be unable to follow the injected instructions, and the failure of encoded payloads may likewise reflect limited decoding ability, as Karlsen et al. [3] also observed for one of their tests. The results therefore show that

capability and susceptibility are confounded, not that scale is itself a weakness.

In a training-data disclosure test, models fine-tuned on company-like employee data leaked at 78.57 % accuracy (average similarity 0.8401) for GPT-4o Mini and 75 % (0.7985) for GPT-2, judged by a finetuned GPT-4o Mini evaluator and embedding similarity with a 0.75 threshold; the number of queries is not reported. In an unbounded-consumption test on a mid-range GPU machine, the average response time of a pretrained model (named as GPT-2 in the abstract) rose from 94.68 s at 10 concurrent requests to more than 35,000 s at 10,000, and that of a model trained from scratch rose from 0.82 s to 489.61 s. The text and the figure labels for the 10,000-request value appear to differ, and the conclusion's "10x increase" does not match these numbers, so the original should be checked before the figure is quoted.

7. Quantitative Security Assurance and Guardrail Requirements

Karlsen et al. [3] propose an assurance metric $AM = RM - VM$, where RM scores fulfilment of security requirements (1, 0.5 or 0 per control question, weighted by importance) and VM scores vulnerabilities found by tests, weighted by a CVSS v4.0 severity. Requirements are grouped into input validation, output validation, training data, development and deployment, access control, third-party services and security procedures; vulnerability tests follow the OWASP list. The normalised score is mapped to five assurance levels. A script (VulnScanLLM.py) automates the tests, and for comparison the scanner Garak was run on one model (more than four hours, compared with under a minute for the script).

In four cases, Llama2-uncensored scored lowest (VM 36.525, normalised score 0.36, low security), Llama2 higher (21.735, 0.62), and Deepseek-coder (10, 0.82) and Samantha-Mistral (7.95, 0.86) highest in the vulnerability-based ranking. The full AM results differ: Case 1 scored 0.35 (low), Case 2 0.46 and Case 0.56 (both moderate), and Case 4, which also had better design requirements (RM 48.2), scored 0.77 (high). Consequently Deepseek-coder is rated high

in one table and moderate in the other because the scoring basis differs. The authors acknowledge that the requirement scores are hypothetical and that tests used a fixed temperature and seed with one to a few tests per technique.

The prevention strategies that [3] lists from OWASP map directly onto guardrail layers: privilege control, segregating external content from prompts and trust boundaries between the model, external sources and functions (LLM01); treating the model as an untrusted user, validating and sanitising inputs and encoding outputs (LLM02); limiting tools and permissions, authorisation in downstream systems and human-in-theloop control (LLM08); and logging, monitoring, rate limiting and patching as security procedures. Their common principle is that the application, not the model, must enforce security decisions. Kurniawan and Chandra [5] also point to spotlighting [10], a defence proposed to mitigate indirect prompt injection, among the measures in the wider literature.

8. Synthesis

Four patterns emerge. First, detectors fall into three families that trade accuracy against cost: scanners [4] are cheap but weakly evidenced, reasoning-based detection [1] and mutation-based detection [2] reach higher reported accuracy but add inference time or extra model queries. Second, no study uses the same dataset or baselines as another, so reported accuracies are not directly comparable. Third, detection is evaluated on prompts, while the highest-impact failures described in [3] concern actions and outputs. Fourth, assurance frameworks such as [3] can organise the evidence, but their scores depend on weights and test counts that remain small.

IV. COMPARATIVE ANALYSIS

Table 1 compares the five primary studies. Reported metrics are not directly comparable because tasks, datasets, models and evaluation protocols differ.

Table 1. Comparison of the reviewed studies.

Study	Objective	Method / technology	Data and models	Major reported findings	Limitations
Shi et al., 2025 [1]	Detect prompt injection	Fine-grained CoT analysis: step and path consistency, confidence, backtracking over a	Deepset + Safe-Guard test sets, 2,176 prompts; Qwen2.5 0.5B–14B, SFT on OpenO1-SFT	14B: 93.20 % accuracy, 90.13 % F1; 1.16 points accuracy and 3.39 points F1 above LLM Guard	Latency from about 1 s to several or more than 10 s; text only; validation overlap
Zhang et al., 2025 [2]	Universal detection of jailbreak and hijack attacks, text and image	Input mutation and KL divergence of response similarity; 18 mutators, combination policy	11,000 samples, 15 attack types; GPT3.5-Turbo-1106, MiniGPT-4; 12 baselines	Text 86.14 %, image 82.90 % accuracy; best baselines 74.33 % and 70.70 %	Extra model queries; per-model tuning; false positives on open-ended benign prompts;
Karlsen et al., 2026 [3]	Quantitative assurance of LLM applications	AM = RM – VM, CVSS v4.0 weights, OWASP-aligned tests, five levels	Four cases: Llama2uncensored, Llama2, Deepseek-coder, Samantha-Mistral	Uncensored model lowest; fine-tuned and censored models score higher	Hypothetical requirement scores; few tests per technique; scoring bases differ

Pokhrel et al., 2024 [4]	Evaluate an open-source scanner	Vigil: vector search, YARA rules, ML classifier, score aggregation	200 GPT-3.5 generated prompts	178/200 correct (89 %)	Small synthetic data; no comparison; text and tables
Kurniawan & Chandra, 2025 [5]	Simulate injection, data disclosure and resource	Automated attack scripts; fuzzy-match success; embedding based	GPT-2, GPT-3.5, GPT-4o Mini; 20 trials per technique	20/20 plaintext success on GPT-4o Mini for three techniques;	Capability confounded with susceptibility; sample size of

V. PROPOSED CONCEPTUAL FRAMEWORK

detection as one layer among several, and every stage is intended to be enforced by the application rather than by the model.

Fig. 1 presents a conceptual framework derived from the reviewed literature and from the layered concept in the authors' own earlier draft; it has not been implemented or experimentally validated. It treats

Fig. 1. Layered Real-Time Prompt Injection Detection and Guardrails for LLM Applications (conceptual, not experimentally validated).

Stage	Function	Supporting evidence in the reviewed studies
1. Input collection	Receive the user request and any external content	Sources of untrusted content: [3], [7]
2. Source identification	Label each input as authorised instruction, user input, upload, retrieved content or tool output	Segregation of external content and trust boundaries [3]
3. Input and context screening	Rules, classifiers, scanners, reasoning- or mutation-based detection	[1], [2], [4]
4. Risk assessment	Score risk; continue, restrict, request human review or reject	Thresholds $\delta = 0.37$ [1], $\theta = 0.02 / 0.025$ [2], threatscore aggregation [4]
5. Secure context construction	Separate instructions from untrusted data; pass only needed information	Separation alone gave limited detection [2]
6. LLM processing	Generate the response	—
7. Output validation	Check format, sensitive content and policy; encode output	OWASP LLM02 mitigations [3]
8. Action-level control	Least privilege, validated tool calls, human approval for high-impact actions	OWASP LLM08 mitigations [3]

A cross-cutting layer provides logging, monitoring, rate limiting and patching [3], [5]. Stages 1–5 can be tiered: inexpensive rules or scanners run first, and costlier reasoning- or mutation-based detection runs only on suspicious inputs, to contain the latency and query overhead reported in [1] and [2]. Output

validation cannot undo an action already taken, so stage 8 must run before execution. The total effect of the combined layers must be measured by testing; it is not established by the reviewed studies.

VI. RESEARCH GAPS AND FUTURE DIRECTIONS

Table 2. Research gaps and future directions.

Gap supported by evidence	Supporting studies	Future direction
Small, synthetic or single-model evaluations	[3], [4], [5]	Larger shared benchmarks with real-world and multimodel data
Non-comparable datasets, baselines and metrics	[1], [2], [4]	Common benchmark reporting accuracy, precision, recall, F1, missed-attack rate and
Latency and query cost rarely controlled	[1], [2], [3], [5]	Report latency and cost; tiered and low-budget detection
False positives on open-ended or sensitivesounding benign prompts	[1], [2]	Dedicated benign hard-case sets; calibrated thresholds
Adaptive attackers tested only in part	[2]	Adaptive red-teaming of each detector, not only of one component
Detection and action-level controls not evaluated together in agents	[3], [5]	End-to-end tests of tool-calling agents with least-privilege and approval gates
Assurance scores sensitive to weights and test counts	[3]	Calibrated weights, more tests, consistent scoring basis
Mostly text and image; other modalities and long contexts untested	[1], [2]	Extend detection to audio, documents and multi-agent settings

Directions still requiring investigation include standardised reporting of safety-case evidence, humanfactors studies of approval fatigue in human-in-the-loop controls, and cost-benefit analysis of layered defences.

VII. CONCLUSION

Research on prompt-injection defence has moved from keyword filtering towards reasoning-based, mutation-based and scanner-based detectors, together with assurance frameworks that organise guardrail requirements. Reported accuracy is promising for specific tasks, but evidence is dominated by small, synthetic or single-model evaluations, comparisons across studies are not possible, and the cost of detection is seldom reported. Because no detector is complete, reliable deployment requires layered defences in which the application enforces permissions and validates outputs and actions, supported by evaluations that include adaptive attackers and agentic settings, as reflected in the proposed framework.

REFERENCES

1. L. Shi, Y. Kang, J. Hu, X. Li, and M. Yang, "Meticulous Thought Defender: Fine-grained chain-of-thought (CoT) for detecting prompt injection attacks of large language models," IEEE Access, vol. 13, pp. 113194– 113207, 2025, doi: 10.1109/ACCESS.2025.3583759.
2. X. Zhang, C. Zhang, T. Li, Y. Huang, X. Jia, M. Hu, J. Zhang, Y. Liu, S. Ma, and C. Shen, "JailGuard: A universal detection framework for prompt-based attacks on LLM systems," ACM Trans. Softw. Eng. Methodol., vol. 35, no. 1, Art. 8, Dec. 2025, doi: 10.1145/3724393.
3. S. S. Karlsen, M. M. Yamin, E. Hashmi, B. Katt, and M. Ullah, "Securing large language models: A quantitative assurance framework approach," J. Inf. Secur. Appl., vol. 97, Art. 104351, 2026, doi: 10.1016/j.jisa.2025.104351.
4. K. Pokhrel, C. Sanin, M. K. H. Sakib, M. R. Islam, and E. Szczerbicki, "An adversarial machine learning approach on securing large language model with Vigil, an open-source initiative,"

- Procedia Comput. Sci., vol. 246, pp. 686–695, 2024, doi: 10.1016/j.procs.2024.09.486.
5. A. Kurniawan and M. B. Chandra, "Simulation of prompt injection attacks on generative pre-trained transformers models," *Procedia Comput. Sci.*, vol. 269, pp. 400–410, 2025, doi: 10.1016/j.procs.2025.08.292.
 6. Y. Liu et al., "Prompt injection attack against LLM-integrated applications," *arXiv:2306.05499*, 2023.
 7. K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, "Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection," in *Proc. 16th ACM Workshop Artif. Intell. Secur.*, 2023, pp. 79–90, doi: 10.1145/3605764.3623985.
 8. F. Perez and I. Ribeiro, "Ignore previous prompt: Attack techniques for language models," *arXiv:2211.09527*, 2022.
 9. A. Zou, Z. Wang, N. Carlini, M. Nasr, J. Z. Kolter, and M. Fredrikson, "Universal and transferable adversarial attacks on aligned language models," *arXiv:2307.15043*, 2023.
 10. K. Hines et al., "Defending against indirect prompt injection attacks with spotlighting," *arXiv:2403.14720*, 2024.
 11. OWASP, "OWASP Top 10 for LLM Applications," 2023. [Online]. Available: <https://genai.owasp.org/llm-top10/>
 12. A. Robey, E. Wong, H. Hassani, and G. J. Pappas, "SmoothLLM: Defending large language models against jailbreaking attacks," *arXiv:2310.03684*, 2023.
 13. Vigil, "Vigil: Documentation," 2024. [Online]. Available: <https://vigil.deadbits.ai/>