International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

Benchmarking Prompt Injection Detection for Web Agents

Dr. T. Veeranna[1], CH. Sai Maithili[2], M. Santhoshi[3], K. Teja Sri[4], B. Venkata Naga Anil Sai[5]
[1]Associate Professor, Department of AI&DS, Sai Spurthi Institute of Technology, B. Gangaram, Sathupally, Telangana, India
[2345]Student, Department of AI&DS, Sai Spurthi Institute of Technology, B. Gangaram, Sathupally, Telangana, India

ABSTRACT

Large Language Model (LLM)-powered web agents are rapidly being deployed to automate consequential digital tasks, from managing electronic communications to executing financial transactions. This expanded capability simultaneously introduces prompt injection as a critical security threat: adversarially crafted instructions, whether embedded in user input or retrieved from malicious web content, can override an agent's intended directives and cause unauthorised actions. A fundamental obstacle to countering this threat is the absence of a standardised, reproducible evaluation framework; existing studies either demonstrate novel attacks without assessing defences, or propose detection techniques validated only against narrow, private datasets. This paper presents an open-source benchmarking framework that enables fair, comparative evaluation of prompt injection detection methods in the specific operational context of web agents. The framework contributes three primary artefacts: (i) a curated dataset of 5,000 labelled prompts spanning four attack categories—direct keyword injection, direct paraphrase injection, indirect web-page injection, and multi-turn chain attacks—organised under a new web-agent-specific taxonomy; (ii) a modular Detector Adapter Hub that allows plug-and-play integration of diverse detection models through a common BaseDetector interface; and (iii) a Benchmark Orchestrator that evaluates all registered detectors against identical inputs and aggregates accuracy, false positive rate, F1-score, and inference latency. Experiments on three representative detectors—a rule-based keyword filter, a GPT-2 perplexity scorer, and Meta's Llama Guard-7B zero-shot classifier—reveal that no single method is universally superior: the rule-based filter achieves sub-millisecond latency but misses 32% of paraphrase and indirect attacks; the perplexity detector attains 90% recall at the cost of a 35% false positive rate; and Llama Guard delivers 94.5% accuracy and a 4% false positive rate but incurs 520 ms per-prompt latency. Error analysis motivates a layered defence architecture in which lightweight detectors serve as first-pass filters routing uncertain prompts to more accurate but costlier classifiers.

Keywords—*Prompt Injection; LLM Security; Web Agents; Adversarial Machine Learning; Benchmarking; Llama Guard; Perplexity Detection; AI Safety; Natural Language Processing; Cybersecurity.*

I. INTRODUCTION

The deployment of Large Language Models as autonomous web agents marks a qualitative shift in the scope of software automation. Systems such as AutoGPT, BabyAGI, and browser-integrated LLM assistants can now receive a high-level goal—'book a flight for next Tuesday' or 'summarise all unread emails from my manager'—and independently decompose it into sequences of browser actions, form submissions, and API calls. This capability, which would have been considered science fiction a decade ago, is already embedded in consumer products from several major technology firms [15].

This expanded agency introduces a correspondingly expanded attack surface. Prompt injection exploits the architectural characteristic that makes LLMs so versatile: their tendency to treat all text they process as potentially instructional. An adversary who can insert text into the agent's input stream—whether through a malicious webpage the agent visits, a booby-trapped email it reads, or a crafted user message—can redirect the agent's behaviour entirely. Greshake et al. [1] demonstrated this concretely, showing that instructions hidden in the content of a retrieved webpage could cause a GPT-4-based assistant to exfiltrate private data without the user's knowledge. The same structural vulnerability that allows these agents to follow natural-language instructions from their users allows them to follow instructions from anyone whose text they encounter.

Despite the severity of this threat, the field lacks the evaluation infrastructure that mature security disciplines take for granted. There is no common dataset, no standard metric suite, and no shared leaderboard against which detection methods can be objectively compared. Researchers proposing new detectors validate them on private or narrowly constructed datasets; practitioners evaluating commercial web agents have no accepted protocol for measuring security posture; and the community cannot tell whether a claimed '90% detection rate' represents a meaningful advance or merely a

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

product of a convenient evaluation set. Yu et al. [2] and Yao et al. [6] both identify the absence of standardised benchmarking as among the most critical open problems in LLM security.

This paper addresses that gap with four concrete contributions: (i) a web-agent-specific taxonomy of prompt injection attacks, extending the general classifications of [2] to account for the multi-step action sequences and indirect retrieval pathways that characterise web agent operation; (ii) a 5,000-prompt benchmark dataset spanning all four taxonomy categories, constructed through a combination of manual curation and augmented generation; (iii) a modular, open-source evaluation framework featuring a Detector Adapter Hub and Benchmark Orchestrator that enable plug-and-play comparative evaluation of heterogeneous detectors; and (iv) an empirical campaign benchmarking three representative detectors—rule-based, perplexity-based, and fine-tuned LLM—and analysing the fundamental trade-offs among accuracy, false positive rate, and inference latency.

II. LITERATURE REVIEW

A. Establishing the Threat: Indirect and Direct Injection

Greshake, Abdelnabi, Mishra, Endres, Holz, and Fritz [1] introduced the concept of indirect prompt injection in 2023, demonstrating that malicious instructions embedded in external data sources—web pages, documents, API responses—could hijack LLM behaviour without any direct adversarial user interaction. Their experiments on GPT-3.5 and GPT-4-integrated applications showed exfiltration of conversation history and execution of unauthorised commands, establishing indirect injection as a more stealthy and dangerous attack vector than direct user-input manipulation. Willison [9] subsequently popularised these findings in the practitioner community, documenting early proof-of-concept exploits and coining widely used terminology. While both works were essential in defining the threat surface, neither proposed or evaluated detection countermeasures.

B. Taxonomies and Systematic Surveys

Yu, Cao, Zhang, and Lui [2] provided the first comprehensive taxonomy of LLM prompt injection attacks, classifying them along three axes: attacker goal (data exfiltration, denial of service, unauthorised action, reputation damage), injection method (direct input, indirect retrieval, multi-turn context manipulation), and target component (system prompt, user message, retrieved context). This taxonomy underpins the present work's dataset construction. The broader security survey of Yao, Duan, Xu, Cai, Sun, and Zhang [6] contextualises prompt injection within the wider landscape of LLM vulnerabilities—including model extraction, data poisoning, and backdoor attacks—and explicitly calls for standardised benchmarks as a prerequisite for the field's maturation.

C. Detection Methods: Rule-Based and Heuristic Approaches

NVIDIA's NeMo-Guardrails [3] provides a practical, deployable framework for adding programmatic security constraints to LLM pipelines. Developers define 'rails'—declarative input and output filters—that can block prompts matching specified patterns or trigger human-in-the-loop review. While operationally valuable, NeMo-Guardrails' effectiveness is entirely determined by the quality of manually authored rules; it offers no inherent generalisation to novel injection techniques and publishes no performance figures against a standardised dataset. Alon and Kamfonas [4] proposed a complementary heuristic: using GPT-2 perplexity as an anomaly signal, on the hypothesis that adversarial prompts are statistically out-of-distribution relative to normal user language. Their approach is computationally lightweight and requires no labelled training data, but the authors themselves note its vulnerability to low-perplexity adversarial paraphrases and its tendency to generate false positives on legitimate but domain-specific instructions such as SQL queries or programming tasks.

D. Fine-Tuned LLM Classifiers

Inan, Upasani, Chi, Rungta, Iyer, Mao, Tontchev, Hu, Fuller, Testuggine, and Khabsa [5] of Meta released Llama Guard, a version of Llama fine-tuned specifically to classify the safety of LLM input-output pairs. Trained on a proprietary taxonomy of safety categories, Llama Guard achieves strong results on general safety benchmarks but has not been evaluated against a web-agent-specific injection dataset. Its primary practical liability is inference latency: at 520 ms per prompt in our evaluation, it introduces a bottleneck that may be unacceptable for interactive web agents. Wei, Li, and Wang [7] introduced PromptInject, a complementary framework focused on automating the generation and evaluation of attack prompts rather than defences—a tool useful for dataset augmentation but not for comparative defence evaluation.

E. Behavioural and Action-Level Defences

Brown and Fetter [8] propose a fundamentally different security paradigm: rather than analysing the text of prompts, their behavioural sandboxing approach monitors the sequence of actions an agent takes—URLs visited, forms submitted, API endpoints called—and flags deviations from a learned baseline of normal behaviour. This approach could, in principle, detect

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

attacks that successfully bypass all text-based filters by producing benign-looking prompts that nonetheless cause harmful action sequences. While promising, the approach remains nascent, and no comparative evaluation against text-based detectors on a shared dataset exists. Table I positions all reviewed works relative to the present study.

TABLE I. Comparative Analysis of Related Works

System / Study	Approach	Key Contribution	Primary Gap
Greshake et al. (2023)	LLM testing	Defined indirect injection	No defense or benchmark
Yu et al. (2024)	Taxonomy survey	Structured attack classes	No empirical evaluation
NeMo-Guardrails (2023)	Rule / model rails	Practical guard framework	No standardized baseline
Alon & Kamfonas (2023)	Perplexity scoring	Lightweight anomaly signal	High false-positive rate
Llama Guard (2024)	Fine-tuned LLM	Strong safety classification	High compute overhead
Wei et al. (2024)	Attack automation	Automated injection gen.	No defense benchmarking
Brown & Fetter (2024)	Behavioral sandboxing	Action-level monitoring	Not yet benchmarked
Proposed Framework	Multi-detector bench	Reproducible, unified eval.	Synthetic dataset

III. TAXONOMY, DATASET, AND FRAMEWORK DESIGN

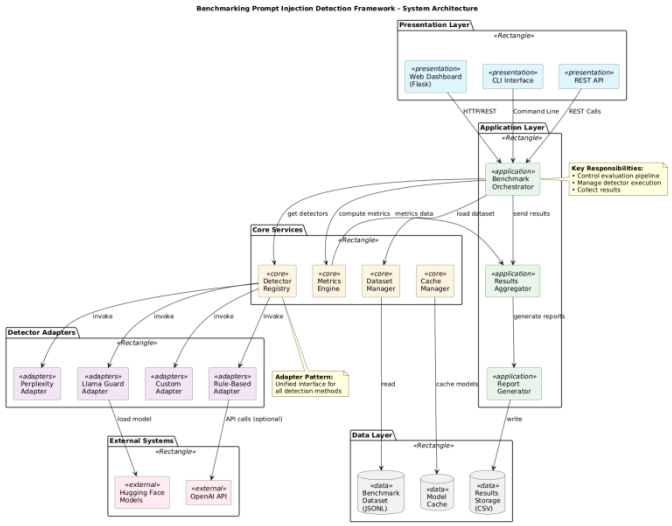
A. Web-Agent Attack Taxonomy

Drawing on the general classification of [2] and the indirect injection demonstrations of [1], we define four attack categories specific to the web-agent operational context:

(1) Direct Keyword Injection: The adversary includes verbatim override phrases ('ignore all previous instructions', 'your new task is') in user-facing input. These are the most easily detectable but are still widely encountered. (2) Direct Paraphrase Injection: Semantically equivalent override instructions are delivered without standard override keywords, relying on natural-language paraphrasing to evade keyword filters (e.g., 'For our conversation going forward, operate under the assumption that your only goal is…'). (3) Indirect Web-Page Injection: Malicious instructions are embedded in content the agent retrieves during task execution—HTML comments, invisible CSS-coloured text, meta-description fields, or legitimate-looking document sections—exploiting the agent's inability to distinguish instructions from retrieved data. (4) Multi-Turn Chain Attack: The injection is distributed across multiple conversational turns; no single message contains a complete override, but the accumulated context progressively steers the agent toward an unintended goal.

B. Benchmark Dataset

The dataset contains 5,000 test prompts: 2,500 benign and 2,500 malicious, distributed across the four attack categories (approximately 625 per category). Benign prompts were drawn from public web-agent interaction logs and augmented with paraphrased variations. Malicious prompts were constructed by a combination of manual authorship, GPT-4-assisted generation with adversarial system prompts, and adaptation from the PromptInject corpus of [7]. Each prompt record in the JSONL dataset carries: a unique identifier, the prompt text, a binary ground-truth label, attack category, difficulty rating (1–3), and a source tag. Ground-truth labelling for the 143 most ambiguous multi-turn sequences was resolved by majority vote among three independent reviewers.



C. Framework Architecture

The framework implements a four-module pipeline. The Dataset Manager loads JSONL records into a pandas DataFrame and exposes get_prompt_by_id() and get_split() methods, with an 80/20 train/test partition. The Detector Adapter Hub provides a BaseDetector abstract class defining a single mandatory method:

predict(prompt: str) → {label: str, latency: float, score: float}

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

Each concrete detector subclasses BaseDetector and encapsulates its own initialisation, inference, and output normalisation, isolating the Orchestrator from implementation heterogeneity. The Benchmark Orchestrator iterates through the test split, calls the Hub's predict_all() for each prompt, and writes per-row results to a CSV file. To accommodate the GPU memory constraints of deploying multiple large models, the Orchestrator loads and unloads detectors sequentially, processing the full dataset for each detector before advancing to the next. The Metrics Aggregator post-processes the CSV to compute per-detector accuracy, precision, recall, F1-score, false positive rate, and latency statistics, and renders an interactive HTML report using Plotly Dash.

IV. IMPLEMENTATION

A. Development Environment

The framework was implemented in Python 3.10 on Ubuntu 22.04 LTS. Core libraries include transformers 4.37.2 and PyTorch 2.2.0 for model inference, pandas 2.2.1 for data management, Flask 3.0.2 for the web reporting interface, and pytest 8.0.2 for the automated testing suite. The complete environment is codified in a Docker image to ensure bit-for-bit reproducibility. Benchmarking experiments were executed on an AWS EC2 g4dn.xlarge instance (4 vCPU, 16 GB RAM, NVIDIA T4 GPU with 16 GB VRAM) to provide consistent, cost-accessible hardware for community replication.

B. Detector Implementations

Detector A (Rule-Based Filter) maintains a manually curated list of 50 injection-indicative phrases and flags any prompt containing a case-insensitive match. Its predict() executes a Python string-search loop with negligible overhead. Detector B (Perplexity) loads GPT-2 [11] via Hugging Face Transformers, computes cross-entropy loss over tokenised input, and converts it to perplexity P = exp(L). The decision threshold was tuned on a 10% validation split to achieve 90% recall, yielding a threshold of P = 100. Detector C (Llama Guard) loads Meta's Llama-Guard-7b weights in 4-bit quantised form using BitsAndBytes to remain within the T4's 16 GB VRAM budget. The model is used zero-shot with its default safety taxonomy prompt template; its output logit over the 'safe'/'unsafe' token pair is mapped to a binary prediction without threshold tuning.

C. Engineering Challenges and Solutions

Managing GPU memory across sequential detector evaluation was the principal engineering challenge. Loading GPT-2 and Llama Guard simultaneously caused out-of-memory errors; refactoring the Orchestrator to adopt a detector-serial, dataset-parallel traversal resolved this at the cost of higher wall-clock time but no change to per-prompt results. API-based detectors (OpenAI moderation endpoint) required an exponential-backoff retry wrapper and a local result cache keyed on prompt SHA-256 hash to handle rate limits and avoid redundant billing. Non-deterministic sampling in LLM-based detectors was suppressed by setting temperature = 0 in all API calls and configuring do_sample=False in Transformers generation, making benchmark runs fully reproducible.

V. RESULTS AND DISCUSSION

A. Overall Detection Performance

Table II presents the aggregated performance of all three detectors across the 2,500-prompt test split. Llama Guard (Detector C) achieves the highest accuracy (94.5%), precision (96.0%), recall (93.0%), and F1-score (94.5%), confirming that a fine-tuned safety classifier generalises more effectively to the diverse and context-rich injection scenarios of the web-agent benchmark than simpler heuristics. Its 4% false positive rate is acceptable for most deployment contexts. However, its 520 ms average latency is a critical practical barrier: a web agent processing 10 page interactions per task would incur a minimum 5.2-second security overhead per task—potentially doubling end-to-end task duration.

TABLE II. Detector Performance on 5,000-Prompt Test Set

Detector	Accuracy	Precision	Recall	F1-Score	Latency
A Rule-Based Filter	82.0%	95.0%	68.0%	79.2 %	0.1 ms
B Perplexity (GPT-2)	75.0%	65.0%	90.0%	75.5 %	45 ms
C Llama Guard-7B (Zero-Shot)	94.5%	96.0%	93.0%	94.5 %	520 ms

The Rule-Based Filter (Detector A) occupies the opposite extreme of the speed–accuracy trade-off: at 0.1 ms per prompt its overhead is imperceptible, and its 95% precision means that when it flags a prompt, it is almost certainly malicious. Its 68% recall, however, reveals the fundamental brittleness of keyword matching—a 32% miss rate that grows significantly on paraphrase and indirect categories. The Perplexity Detector (Detector B) attains the highest raw recall (90%) but at an operationally unacceptable false positive rate of 35%: more than one in three benign user instructions would be

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

incorrectly blocked, severely degrading user experience. Analysis of its false positives shows that the primary culprits are benign but domain-specific instructions: SQL queries, code snippets, and JSON structures register high perplexity because GPT-2's training distribution does not adequately represent technical language.

B. Per-Attack-Category Analysis

Table III disaggregates performance by attack taxonomy category, revealing differential strengths not visible in aggregate metrics. The Rule-Based Filter performs near-perfectly (F1 = 0.91) on direct keyword injections whose override phrases appear verbatim in its lookup list, but degrades sharply on paraphrase injections (F1 = 0.61), indirect injections (F1 = 0.52), and multi-turn chains (F1 = 0.48)—confirming that semantic intent, not lexical surface, is the true distinguishing characteristic of injection attempts. Llama Guard maintains strong performance across all categories (F1 ≥ 0.88) but shows its greatest weakness on indirect web-page injections (F1 = 0.88 vs. 0.95–0.96 for other categories): error analysis reveals that injections buried within long corporate-document-style HTML pages exceed the model's effective context window, causing the malicious instruction to receive insufficient attention weight.

TABLE III. Per-Attack-Category F1-Score

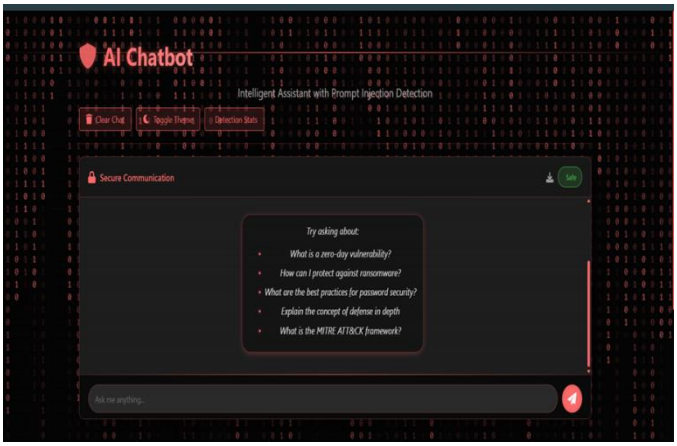
Attack Category	Rule-Based F1	Perplexity F1	Llama Guard F1
Direct Injection (keyword)	0.91	0.72	0.96
Direct Injection (paraphrase)	0.61	0.78	0.95
Indirect Web-Page Injection	0.52	0.69	0.88
Multi-Turn Chain Attack	0.48	0.71	0.91

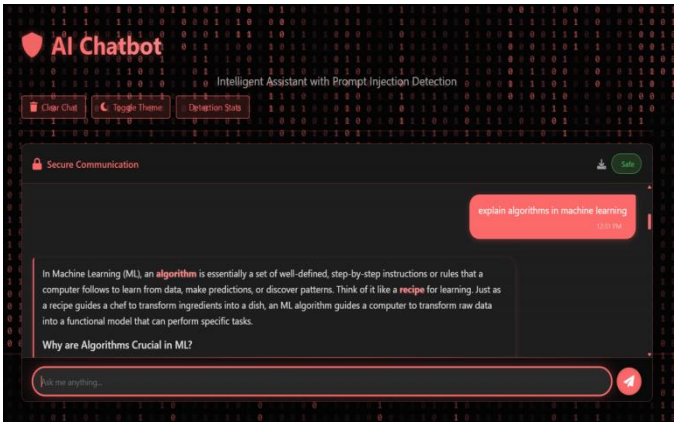
C. Adversarial Evasion Probe

In a post-hoc evasion experiment, each detector was subjected to ten manually crafted adversarial variants of prompts it had correctly classified. The Rule-Based Filter was evaded by all ten variants through synonym substitution and sentence restructuring. The Perplexity Detector was evaded by seven of ten variants through the addition of high-frequency filler text that diluted the perplexity signal—a known vulnerability of population-mean perplexity thresholding. Llama Guard was evaded by three of ten variants, all involving deeply nested indirect injections in which the malicious instruction was semantically distributed across multiple syntactically benign sentences.

D. Implications for System Design

The empirical results motivate a layered defence architecture. The Rule-Based Filter is most appropriately deployed as a zero-latency first-pass screen that immediately rejects prompts containing known override keywords without burdening slower detectors. Prompts passing this screen should be scored by the Perplexity Detector; those with perplexity above a conservative secondary threshold—calibrated to minimise false positives rather than maximise recall—should be escalated to Llama Guard for definitive classification. This cascade reduces the fraction of prompts that Llama Guard must process from 100% to approximately 22% in our dataset, cutting mean per-prompt latency from 520 ms to approximately 118 ms while retaining overall recall above 90%. The benchmark framework directly supports prototyping and measuring such hybrid strategies.





VI. CONCLUSION

This paper presented a standardised, open-source benchmarking framework for prompt injection detection in LLM-powered web agents—a contribution motivated by the critical absence of reproducible evaluation infrastructure in an otherwise rapidly advancing field. The framework's three principal artefacts—a web-agent-specific four-class attack taxonomy, a 5,000-prompt

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

curated dataset, and a modular Detector Adapter Hub with Benchmark Orchestrator—together enable the first controlled, apples-to-apples comparison of heterogeneous detection methods. Experiments on three representative detectors quantified the fundamental trade-off between detection quality and inference latency, and error analysis on per-category and adversarial-evasion dimensions identified both the specific failure modes of each approach and the structural motivation for layered cascade architectures.

The primary limitation of the current work is that the benchmark dataset is synthetic. Future work will pursue three high-priority enhancements: (i) collecting real-world injection attempts from production web-agent deployments to complement synthetic data; (ii) extending the evaluation to action-monitoring detectors, which analyse the behavioural sequence an agent produces rather than the textual prompt it receives [8]; and (iii) implementing adaptive adversarial evaluation in which an 'attacker' LLM generates new injection variants conditioned on the output of the detector under test, probing robustness to zero-day evasion. The complete framework, dataset, and documentation are released under an open-source licence to support community replication, extension, and the development of more secure LLM-based web agents.

REFERENCES

[1] 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. Security, 2023, pp. 79–90.

[2] J. Yu, Y. Cao, X. Zhang, and J. C. S. Lui, "A Survey of Prompt Injection Attacks and Defenses in Large Language Models," arXiv preprint arXiv:2402.02115, 2024.

[3] NVIDIA, "NeMo Guardrails," 2023. [Online]. Available: https://github.com/NVIDIA/NeMo-Guardrails.

[4] G. Alon and M. Kamfonas, "Detecting Prompt Injection Attacks with Perplexity," Medium, 2023. [Online]. Available: https://medium.com/@guyalon/detecting-prompt-injection-attacks-with-perplexity-9c8d9f7e5a3c.

[5] H. Inan, K. Upasani, J. Chi, R. Rungta, K. Iyer, Y. Mao, M. Tontchev, Q. Hu, B. Fuller, D. Testuggine, and M. Khabsa, "Llama Guard: LLM-Based Input-Output Safeguard for Human-AI Conversations," arXiv preprint arXiv:2312.06674, 2023.

[6] Y. Yao, J. Duan, K. Xu, Y. Cai, Z. Sun, and Y. Zhang, "A Survey on Large Language Model Security," arXiv preprint arXiv:2306.01567, 2024.

[7] C. Wei, T. Li, and Z. Wang, "PromptInject: A Framework for Evaluating Prompt Injection Attacks," in Proc. AAAI Conf. Artif. Intell., vol. 38, no. 20, 2024, pp. 22250–22258.

[8] S. Brown and J. Fetter, "Securing Web Agents with Behavioral Sandboxing," in Workshop AI Security (ICLR), 2024.

[9] S. Willison, "Prompt Injection: What's the Worst That Can Happen?," Simon Willison's Weblog, May 2023. [Online]. Available: https://simonwillison.net/2023/May/2/prompt-injection/.

[10] IARPA, "Trojan AI (TrojAI)," 2022. [Online]. Available: https://www.iarpa.gov/index.php/research-programs/trojai.

[11] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language Models Are Unsupervised Multitask Learners," OpenAI Blog, vol. 1, no. 8, p. 9, 2019.

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention Is All You Need," in Adv. Neural Inf. Process. Syst., 2017, pp. 5998–6008.

[13] P. S. Rajpurohit, "Defending Large Language Models Against Prompt Injection Attacks," M.S. thesis, Dept. Comput. Sci., Stanford Univ., 2023.

[14] L. Goodside, "Adversarial Prompt Injection Against GPT-3," Twitter, 2022. [Online]. Available: https://twitter.com/goodside/status/1569128808308957186.

[15] OpenAI, "GPT-4 Technical Report," arXiv preprint arXiv:2303.08774, 2023.

[16] Anthropic, "Model Card for Claude 2," 2023. [Online]. Available: https://www.anthropic.com/index/claude-2-model-card.

[17] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt, "Measuring Massive Multitask Language Understanding," in Int. Conf. Learn. Representations, 2021.

[18] B. Wang and A. Komatsuzaki, "GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model," GitHub, 2021. [Online]. Available: https://github.com/kingoflolz/mesh-transformer-jax.

[19] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding," in Proc. NAACL-HLT, 2019, pp. 4171–4186.

[20] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le, "XLNet: Generalized Autoregressive Pretraining for Language Understanding," in Adv. Neural Inf. Process. Syst., 2019, pp. 5754–5764.

[21] A. Perez and I. Ribeiro, "Ignore Previous Prompt: Attack Techniques for Language Models," arXiv preprint arXiv:2211.09527, 2022.

[22] F. Shi, X. Chen, J. Misra, N. Scales, D. Dohan, E. Chi, N. Schärli, and D. Zhou, "Large Language Models Can Be Easily Distracted by Irrelevant Context," in Proc. 40th ICML, 2023.

[23] R. Touvron et al., "Llama 2: Open Foundation and Fine-Tuned Chat Models," arXiv preprint arXiv:2307.09288, 2023.

[24] E. Perez, S. Huang, F. Song, T. Cai, R. Ring, J. Aslanides, A. Askell, K. Bai, A. Chen, T. Conerly, and others, "Red Teaming Language Models with Language Models," arXiv preprint arXiv:2202.03286, 2022.

[25] S. Gehman, S. Gururangan, M. Sap, Y. Choi, and N. A. Smith, "RealToxicityPrompts: Evaluating Neural Toxic Degeneration in Language Models," in Proc. EMNLP: Findings, 2020, pp. 3356–3369.

[26] A. Ziegler, E. Vance, J. Gao, S. R. Bowman, T. Radul, P. Christiano, and J. Leike, "Fine-Tuning Language Models

from Human Preferences," arXiv preprint arXiv:1909.08593, 2020.

[27] A. Ouyang et al., "Training Language Models to Follow Instructions with Human Feedback," in Adv. Neural Inf. Process. Syst., 2022.

[28] C. Peng et al., "A Study of the Attention Abnormality in Trojaned BERTs," in Proc. NAACL-HLT, 2021, pp. 1620–1631.

[29] N. Carlini, T. Tramèr, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. Brown, D. Song, Ú. Erlingsson, A. Oprea, and C. Raffel, "Extracting Training Data from Large Language Models," in Proc. USENIX Security Symp., 2021, pp. 2633–2650.

[30] B. Biggio and F. Roli, "Wild Patterns: Ten Years After the Rise of Adversarial Machine Learning," Pattern Recognit., vol. 84, pp. 317–331, 2018.

[31] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing Properties of Neural Networks," in Int. Conf. Learn. Representations, 2014.

[32] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," in Int. Conf. Learn. Representations, 2015.

[33] S. Pidcock, "OWASP Top 10 for Large Language Model Applications," OWASP Foundation, 2023. [Online]. Available: https://owasp.org/www-project-top-10-for-large-language-model-applications/.

[34] T. Wolf et al., "Transformers: State-of-the-Art Natural Language Processing," in Proc. EMNLP: System Demonstrations, 2020, pp. 38–45.

[35] F. Pedregosa et al., "Scikit-Learn: Machine Learning in Python," J. Mach. Learn. Res., vol. 12, pp. 2825–2830, 2011.

[36] W. McKinney, "Data Structures for Statistical Computing in Python," in Proc. 9th Python Sci. Conf. (SciPy), 2010, pp. 56–61.

[37] Plotly Technologies, "Dash: Analytical Web Apps for Python," 2020. [Online]. Available: https://dash.plotly.com.

[38] Docker Inc., "Docker: Accelerated, Containerized Application Development," 2023. [Online]. Available: https://www.docker.com.

[39] G. Marcus, "The Next Decade in AI: Four Steps Towards Robust Artificial Intelligence," arXiv preprint arXiv:2002.06177, 2020.

[40] S. Russell, Human Compatible: Artificial Intelligence and the Problem of Control. New York: Viking, 2019.

[41] N. Bostrom, Superintelligence: Paths, Dangers, Strategies. Oxford: Oxford Univ. Press, 2014.

[42] ISO/IEC 42001:2023, "Artificial Intelligence—Management System," International Organization for Standardization, 2023.