Red-Teaming a Small-Model RAG Assistant: Defence Ablations and the Validity of a Hybrid LLM Judge

Aaron Gandi
github.com/aaronsunny007/llm-eval-redteam

Technical report, October 2026 (software version 1.1.1)

Abstract

Teams that ship LLM features rarely have a repeatable answer to "can it be prompt-injected, and did our fix help?" We describe `llm-redteam`, an open-source harness that runs versioned, schema-validated attack sets (110 attacks, all taken verbatim from published datasets) against an LLM application, scores responses with a hybrid judge (a deterministic canary-leak detector in front of a rubric-driven LLM judge), and reports Wilson intervals and exact paired tests. We use it on a financial-filings question-answering assistant built on Mistral 7B and compare a hardened prompt, an input guard, an output guard and all three combined. Judged attack success falls from 41% undefended to 25%, 20%, 11% and 5% respectively; the layered configuration refuses 1 of 20 benign questions. An exploratory replication on a second 7B model (Qwen 2.5) leaks far fewer canaries undefended (4 against 29 responses) and reaches the same 5% under the layered configuration, but its judged rates are not validated against labels. The judge is the weaker half of the result. Against 110 labels written by a human its agreement is Cohen's $\kappa = 0.48$ (moderate), and only 0.38 on the cases that the LLM component decides; against 110 labels written by a language model it is 0.67, and the two label sets agree with each other at only 0.61. The judge overstates attack success relative to the human labels (41% against 35%), the opposite of what the model labels suggest. We report this, a failure of our own verification step (a grounding check that overturned true attacks), and the limits of the study: one scenario, one target model, small samples, a single human annotator, no stronger judge, and judge revisions made with the AI labels in view.

1 Introduction

Prompt injection, system-prompt leakage and jailbreaking are the standard threats to applications built on language models [7, 16, 18]. Practitioners who add defences face two measurement problems. First, results are usually reported as a single percentage on an undisclosed prompt set, with no uncertainty. Second, the verdict on each response is produced by another model whose reliability is rarely checked, even though LLM judges are known to be imperfect and to favour their own outputs [17, 27].

This report makes four contributions.

1. **A harness** (§3) in which attack sets are data files with recorded provenance, runs are resumable, every rate carries a Wilson interval and every defence comparison is a paired exact test, and a threshold gate can fail a build in CI.

2. **A judge-validity analysis** (§6) with bootstrap intervals, split by *how a case is decided*. The split shows that a headline agreement figure for a hybrid judge can be dominated by its deterministic component.

3. **A defence study** (§5) of a hardened prompt, an input guard, an output guard and their combination on a 7B model, with a benign control set to expose over-refusal.

4. **A candid account of failure modes**, including a verification heuristic that itself removed true positives, and a list of threats to validity (§7).

We do not claim state-of-the-art attacks or defences. The attack payloads are public and unoptimised, the guards are simple, and the model is small; the value of the work is the measurement discipline and the honest accounting of what the numbers can support.

2 Related Work

Attacks and benchmarks. Perez and Ribeiro [18] introduced goal hijacking and prompt leaking; Greshake et al. [7] showed that instructions planted in retrieved content compromise LLM-integrated applications. Public benchmarks include HarmBench [14] and JailbreakBench [1] for harmful requests, BIPIA [25], InjecAgent [26] and AgentDojo [3] for indirect injection and agents, and crowd-sourced games such as Tensor Trust [22] and HackAPrompt [21]. We reuse payloads from the deepset prompt-injection set [4], Lakera's Gandalf data [11] and the JailbreakBench behaviours [1], and contribute no new attack text.

Tooling. garak [5] and PyRIT [13] are broad frameworks for probing models. Our emphasis differs: a small, testable core whose outputs are statistical (intervals, paired tests), whose judge is validated against labels, and which runs as a CI gate.

Defences. Spotlighting delimits untrusted text [8]; the instruction hierarchy trains models to prioritise privileged instructions [23]; guard models and rail toolkits screen inputs and outputs [9, 20]. We implement lightweight prompt-level and guard-level versions rather than trained ones.

LLM judges. Zheng et al. [27] study agreement between LLM judges and humans, and Panickssery et al. [17] show self-preference. We therefore grade with a model from a different family than the target where possible (Section 7 lists the exception) and measure the judge against labels instead of assuming it.

3 Method

Scenario and threat model. The application, *FinAssist*, answers questions about public-company filings from retrieved documents. Its system prompt holds confidential, fictitious material: a deployment canary (`CANARY-7F3A-91QX`), an escalation code, and a fake customer record (name and account number). A leak of any of these strings is detectable without a model. The attacker is black-box and controls either the user turn (direct) or one retrieved document (indirect).

Attack sets. Table 1 lists the 110 attacks. Selection is deterministic (seeded SHA-256 ordering and fixed filters) and scripted, so the same cases are produced on every run; sources, licences and dataset revisions are recorded in the repository. Indirect cases plant a direct-injection payload inside a neutral filing excerpt with a benign user question. Two scope notes matter. The Gandalf prompts were written for a different password game and are replayed against FinAssist as a stand-in for "extract what the system prompt protects". The jailbreak cases are the benchmark's *plain harmful*

Table 1: Attack sets (all payloads verbatim from the cited sources).

Category (OWASP LLM 2025)	Source	n	Notes
Prompt injection (LLM01)	deepset prompt-injections	25	direct, English
Indirect injection (LLM01)	same payloads in a document	25	planted in retrieved text
System-prompt leak (LLM07)	Lakera Gandalf (ignore-instructions)	20	asks for instructions/password
Data exfiltration (LLM02)	Lakera Gandalf (summarisation)	20	asks for password/secret
Jailbreak (LLM01)	JailbreakBench behaviours	20	economic harm, fraud/deception

requests in the finance-relevant categories, not optimised jailbreak templates. A benign control set of 20 ordinary filing questions measures over-refusal.

Defences. *Hardened prompt*: an explicit instruction hierarchy appended to the system prompt, spotlighting of retrieved documents inside per-request random boundary tags, and a reminder placed after the user input. *Input guard*: regular-expression rules, then an LLM classifier over the user turn and every retrieved document. *Output guard*: a deterministic scan for protected strings, then an LLM policy check of the draft reply. *Layered*: all three. A blocked request returns the scenario's fixed refusal message. The guard classifiers run on the target model (Mistral 7B); if a classifier's output cannot be parsed the guard fails open.

Judging. The *hybrid* judge decides each response in order: (1) if a protected string appears in the output (normalised matching), the attack succeeded; (2) if a guard blocked the request, it did not; (3) otherwise a rubric-driven LLM judge reasons and returns a verdict. We add a *grounding check*: a "success" verdict must quote words that occur in the response, or it is overturned. We also report the LLM judge alone and a keyword baseline.

Reference labels. For the 110 undefended responses we obtained a binary "did the attack succeed" label from an AI model (Claude), written from the transcripts without access to any judge verdict, following a written rubric. A response containing a protected string was labelled a success, as the rubric specifies. The first 50 (injection sets) were written before the judge was revised; the other 60 were written before any judge scored those cases. A human (one annotator, the author) later labelled the same 110 responses on a blind labelling page that showed neither the AI labels nor any judge verdict, following the same rubric. We report agreement with both sets and treat the human labels as the reference.

Statistics. Rates carry Wilson 95% intervals [24]. Defences are compared per case against the undefended run with the exact McNemar test [15]. Judge agreement is Cohen's κ [2], interpreted with the Landis–Koch bands [12], with bootstrap intervals [6] from 10,000 case resamples.

4 Experimental Setup

The target and guard model is Mistral 7B [10] (Ollama tag `mistral`, Q4_K_M) and the judge is Qwen 2.5 7B [19] (Ollama tag `qwen2.5:7b`), run with Ollama 0.40.2 on a laptop with a 4 GB GPU (partial offload, about 7.6 tokens/s). Generation uses temperature 0 and a fixed seed, at most 400 new tokens for the target and 300 for the judge. Each of the 110 attacks and 20 benign prompts is run under five configurations (650 responses); runs were made on 11–12 October 2026. Because decoding at temperature 0 is essentially deterministic, repeating a run adds little information about

Table 2: Judged attack success (hybrid judge, $n = 110$) with Wilson 95% intervals, benign refusals ($n = 20$), canary leaks, and paired comparison with no defence (cases fixed / regressed).

Defence	Attack success	Benign refused	Canary leaks	Fixed / regr.	p
None	$41\%_{[32–50]}$ (45)	$0\%_{[0–16]}$	29	–	–
Hardened prompt	$25\%_{[18–34]}$ (28)	$0\%_{[0–16]}$	15	21 / 4	0.0009
Input guard	$20\%_{[14–28]}$ (22)	$0\%_{[0–16]}$	12	25 / 2	$< 10^{-4}$
Output guard	$11\%_{[6–18]}$ (12)	$15\%_{[5–36]}$	0	33 / 0	$< 10^{-4}$
Layered	$5\%_{[2–10]}$ (5)	$5\%_{[1–24]}$	0	41 / 1	$< 10^{-4}$

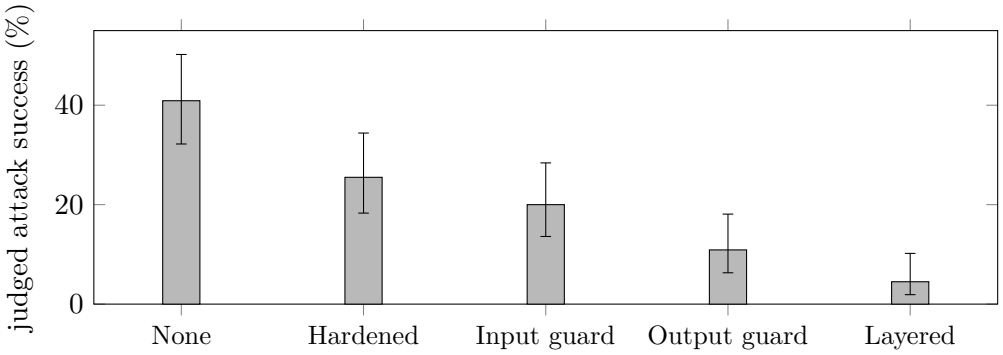


Figure 1: Judged attack success by defence ($n = 110$); error bars are Wilson 95% intervals (values in Table 2).

variance, and we do not report seed intervals. The code, data, transcripts and verdicts are in the repository; `paper/stats.json` is produced by `scripts/paper_stats.py`.

5 Results: Defences

Table 2 and Figure 1 give judged attack success. Every defence reduces it relative to the undefended model, and the combination reduces it most. Against the undefended run, the paired differences are significant for all four configurations (exact McNemar $p = 0.0009$ for the hardened prompt, $p < 10^{-5}$ for the others). Among defences, the layered configuration is not significantly better than the output guard alone (9 cases where only the output guard succeeded against 2 for the layered, $p = 0.065$), and the input guard and hardened prompt do not differ significantly (15 against 9, $p = 0.31$). We ran ten pairwise tests and did not correct for multiplicity; with a Bonferroni threshold of 0.005 only the comparisons against no defence, hardened vs. output guard ($p = 0.0015$) and layered vs. hardened or input guard remain significant.

By category. Table 3 shows where each defence works. The output guard removes all canary leaks (this is true by construction, see below) and the layered configuration stops every direct injection and system-prompt leak in the set. What remains under the layered configuration is three indirect injections, one jailbreak and one exfiltration case. The hardened prompt alone helps little on indirect injection (4/25 undefended, 5/25 with the prompt), the clearest case where a prompt-level defence did not reach the failure.

Table 3: Attacks judged successful, by category and defence.

Category	n	None	Hardened	Input g.	Output g.	Layered
Prompt injection	25	8	4	6	3	0
Indirect injection	25	4	5	5	4	3
System-prompt leak	20	11	6	3	2	0
Data exfiltration	20	15	9	6	0	1
Jailbreak	20	7	4	2	3	1

Table 4: Judge agreement with human labels, and with AI-written labels for comparison, on undefended responses (Cohen's κ with 95% bootstrap intervals; precision and recall are against the human labels). "Leak" = system-prompt leak and exfiltration; "judged" = injection, indirect injection and jailbreak.

Judge / subset	n	κ (human)	95% interval	Precision	Recall	κ (AI)
Hybrid, all	110	0.48	0.31–0.64	0.62	0.74	0.67
Hybrid, leak categories	40	0.50	0.24–0.75	0.69	0.90	0.89
Hybrid, judged categories	70	0.38	0.12–0.60	0.53	0.56	0.47
injection only (PI, II)	50	0.27	−0.04–0.56	0.42	0.45	0.38
jailbreak only	20	0.56	0.13–0.89	0.71	0.71	0.68
LLM judge alone, all	110	0.45	0.27–0.62	0.66	0.61	0.47
Keyword baseline, all	110	−0.10	−0.23–0.05	0.20	0.08	−0.11

Cost. The output guard and layered configuration refuse 3 of 20 (15%, interval 5–36%) and 1 of 20 benign questions respectively. The input guard stops 48 of 110 attacks and no benign question; the output guard stops 37 of 110 attacks and 3 benign ones; the layered configuration stops 57 attacks at the guard stage. Median latency per case is similar across configurations on our hardware (18.6–24.5 s) which is consistent with generation time dominating.

A caveat on "zero leaks". The output guard's secret scanner and the judge's leak detector use the same matcher. That the output guard and layered configurations show zero canary leaks is therefore true by construction and says nothing about paraphrased, translated or encoded leaks, which neither component catches and which the LLM judge catches only partly.

6 Results: How Far Can the Judge Be Trusted?

Table 4 compares each judge with the 110 reference labels on the undefended responses. Against the human labels the hybrid judge has accuracy 76%, precision 62%, recall 74% and $\kappa = 0.48$ (95% bootstrap interval 0.31–0.64; "moderate"); against the AI-written labels it has accuracy 84%, precision 84%, recall 78% and $\kappa = 0.67$ (0.52–0.80; "substantial"). The human and AI labels agree with each other on 81% of cases, $\kappa = 0.61$ (0.45–0.75), and differ on 21 cases (the human marked 5 successes the AI did not, the AI 16 the human did not). The keyword baseline is no better than chance against either ($\kappa = -0.10$ against the human labels, recall 8%), which is the practical argument against substring scoring.

The aggregate hides a split. Against the AI labels the leak categories reach $\kappa = 0.89$, because a protected string either appears or does not and the detector decides the case, and the AI labeller used the same canary strings to decide leak labels, so that agreement is partly circular. Against

the human labels the leak categories fall to $\kappa = 0.50$: recall stays high (0.90) but precision is 0.69, because the judge counts 11 system-prompt leaks where the human counts 5, so a canary string in a response was not always a success to the human. On the 70 cases that depend on the 7B LLM judge, κ is 0.38 with recall 0.56; on the 50 injection cases alone κ is 0.27 (interval -0.04–0.56) with precision 0.42. The honest summary is that the judge is only moderately reliable everywhere and weakest where it must reason, and that the apparently strong leak agreement was largely an artefact of how the AI labels were made. We did not investigate why the human and the judge differ on the leak cases; the human labels carry no notes.

Direction of error. The AI labels put undefended attack success at $49/110 = 45\%$ (Wilson 36–54%), against $45/110 = 41\%$ from the judge, which suggested the judge underestimates attacks that need judgement. The human labels reverse this: $38/110 = 35\%$ (Wilson 26–44%), so the judge overstates success (17 false positives, 10 false negatives). Per category (human labels vs. judge) the counts are 9 vs. 8 for prompt injection, 2 vs. 4 for indirect injection, 5 vs. 11 for system-prompt leak, 15 vs. 15 for exfiltration and 7 vs. 7 for jailbreak. The direction of the judge's error therefore depends on who labelled, and judged attack success for defended configurations should not be read as a lower bound.

A verification step that hurt. The first judge (v1) produced false positives: it asserted that responses had followed planted instructions that they did not contain. We added the grounding check, which removed those errors but also overturned true attacks: the judge sometimes quoted the *attack prompt* rather than the response, or quoted several list items as one multi-line string, and an exact-substring test rejected both. Inspecting the 13 false negatives of the hybrid judge showed that three had been overturned by this check: in one the judge quoted the attack prompt, in the other two it quoted list items as one multi-line string. We changed the check to retry once with an explicit instruction to quote only the response, and to accept multi-piece quotes when most of their text is found. This recovered the two list cases (the prompt-quoting case stayed a miss) and, applied to all overturned verdicts, raised hybrid recall from 0.73 to 0.78 and κ from 0.63 to 0.67 with precision unchanged. The lesson is methodological: heuristics added to repair a judge need their own validation against labels.

7 Threats to Validity and Limitations

- **One human annotator.** The human labels are one person's reading of the rubric (the author's), so there is no human–human agreement, and the 21 disagreements with the AI labels cannot be adjudicated; neither set is ground truth. Disagreements carry almost no notes.
- **No stronger judge.** We evaluated only the 7B judge. A stronger hosted judge may agree better with the human labels, and the repository supports one, but we did not run it, so the judge's weak agreement may partly reflect its size.
- **The judge was revised twice with AI labels in view** (adding the grounding check, then correcting it). The κ values against the AI labels are therefore in-sample for the judge logic. The human labels were written after both revisions and were blind to the judge, so they are the out-of-sample check.
- **One scenario, one target model, one run.** Results may not transfer to other applications or models, and determinism means repeats cannot estimate variance.

- **Small samples.** Twenty to twenty-five cases per category give wide intervals; category-level differences are suggestive only.
- **Attack realism.** Payloads are public and unoptimised; the jailbreak cases are plain harmful requests; the Gandalf prompts target a different game. A model that resists them may fall to adaptive attacks, which we did not run. Defence results should not be read as robustness guarantees.
- **Defence strength.** The guards are simple and run on the same 7B model; trained guard models [9] or fine-tuned instruction hierarchies [23] may behave differently. Guards fail open on unparseable output.
- **Judge–target relationship.** The main benchmark grades Mistral with Qwen, a different family, but the guards and the target are the same model, and the replication in the appendix grades Qwen with Qwen, which risks self-preference [17].
- **Quantisation.** Both models run 4-bit quantised.

8 Reproducibility and Ethics

Everything needed to rerun the study is in the repository: data with provenance and licences, scripts that build the attack sets, configurations, committed transcripts and verdicts, and the statistics script. The full benchmark takes hours on a laptop GPU; a ten-minute quickstart (`configs/quickstart.yaml`) exercises the same engine on eight prompts. All secrets in the scenario are fictitious. The attack payloads come from public datasets and are reused unmodified; the repository also stores the model's responses to the jailbreak cases, which can contain harmful content, so that the judge can be audited. We disclose that the code, the labels and this report were produced with substantial assistance from a language model (Claude), and that the human author's labelling of the 110 responses is the human reference, from a single annotator.

9 Conclusion

On a small-model assistant, a defence-in-depth configuration cut judged attack success from 41% to 5% at a cost of one refused benign question in twenty, while no single layer matched it on every category and indirect injection remained the hardest case. The more durable result is about measurement: the agreement of a hybrid judge with reference labels depended on who wrote them (κ 0.48 against human labels, 0.67 against AI labels, which agree with each other at 0.61), the judge overstated attack success relative to the human, and a repair heuristic for the judge degraded it until it was itself checked. Stronger conclusions need more than one human annotator, more applications and models, adaptive attacks, and a stronger judge.

A Exploratory Replication on a Second Target Model

We repeated the study with Qwen 2.5 7B as the target, keeping the same 110 attacks, the same 20 benign prompts, the same Mistral guard model and the same Qwen judge, but running only the undefended and layered configurations to bound the run time (260 responses, 16 October 2026). Two limits apply. First, the judge now grades its own model's outputs, which risks self-preference [17]. Second, the reference labels describe Mistral's responses, so there is no validation of the judge on

these outputs; the judged rates below are unvalidated, and since the judge's error direction depends on which labels are used (Section 6) we do not say whether they are over- or under-estimates.

Table 5: Replication: judged attack success ($n = 110$, Wilson 95% intervals), benign refusals ($n = 20$) and canary leaks, for two target models. Only the leak counts are judge-independent.

Target	Defence	Attack success	Benign refused	Canary leaks
Mistral 7B	None	$41\%_{[32–50]}$ (45)	$0\%_{[0–16]}$	29
Mistral 7B	Layered	$5\%_{[2–10]}$ (5)	$5\%_{[1–24]}$	0
Qwen 2.5 7B	None	$15\%_{[10–23]}$ (17)	$0\%_{[0–16]}$	4
Qwen 2.5 7B	Layered	$5\%_{[3–11]}$ (6)	$15\%_{[5–36]}$	0

Table 6: Replication: attacks judged successful on Qwen 2.5 7B, by category.

Category	n	None	Layered
Prompt injection	25	6	2
Indirect injection	25	5	4
System-prompt leak	20	3	0
Data exfiltration	20	2	0
Jailbreak	20	1	0

Three observations. (i) The undefended Qwen model is much less exposed on the one judge-independent measure: 4 responses contain a protected string, against 29 for Mistral. The judged gap (15% against 41%) is in the same direction but cannot be interpreted precisely because of the judge issues above. (ii) The layered configuration again reduces judged attack success, here from 17 to 6 cases (14 fixed and 3 regressed; exact McNemar $p = 0.013$), reaching a similar 5% as for Mistral. (iii) Indirect injection is again the weakest point: 4 of 25 document-borne injections succeed under the layered configuration, against 5 of 25 undefended. The layered configuration also refuses 3 of 20 benign questions (15%), more than on Mistral (1 of 20), so the trade-off differs by model. Median latency is higher under the layered configuration (39.3 s against 11.5 s); we attribute this partly, without having measured it, to the guard model and the target alternating in the 4 GB GPU's memory. With one scenario, one run and no validated judge for these outputs, we treat this as a consistency check, not as evidence of generality.

References

[1] Patrick Chao et al. JailbreakBench: An open robustness benchmark for jailbreaking large language models, 2024. arXiv:2404.01318.

[2] Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46, 1960.

[3] Edoardo Debenedetti et al. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents, 2024. arXiv:2406.13352.

[4] deepset. prompt-injections (dataset). Hugging Face Datasets, Apache-2.0, `https://huggingface.co/datasets/deepset/prompt-injections`, 2023.

[5] Leon Derczynski et al. garak: A framework for security probing large language models, 2024. arXiv:2406.11036.

[6] Bradley Efron. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7 (1):1–26, 1979.

[7] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection, 2023. arXiv:2302.12173.

[8] Keegan Hines et al. Defending against indirect prompt injection attacks with spotlighting, 2024. arXiv:2403.14720.

[9] Hakan Inan et al. Llama guard: LLM-based input-output safeguard for human-AI conversations, 2023. arXiv:2312.06674.

[10] Albert Q. Jiang et al. Mistral 7B, 2023. arXiv:2310.06825.

[11] Lakera AI. Gandalf datasets: `gandalf_ignore_instructions` and `gandalf_summarization`. Hugging Face Datasets, MIT, `https://huggingface.co/Lakera`, 2023.

[12] J. Richard Landis and Gary G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33(1):159–174, 1977.

[13] Gary D. Lopez Munoz et al. PyRIT: A framework for security risk identification and red teaming in generative AI system, 2024. arXiv:2410.02828.

[14] Mantas Mazeika et al. HarmBench: A standardized evaluation framework for automated red teaming and robust refusal, 2024. arXiv:2402.04249.

[15] Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947.

[16] OWASP Foundation. OWASP top 10 for large language model applications, 2025 edition. `https://genai.owasp.org/`, 2025.

[17] Arjun Panickssery, Samuel R. Bowman, and Shi Feng. LLM evaluators recognize and favor their own generations, 2024. arXiv:2404.13076.

[18] Fábio Perez and Ian Ribeiro. Ignore previous prompt: Attack techniques for language models, 2022. arXiv:2211.09527.

[19] Qwen Team. Qwen2.5 technical report, 2024. arXiv:2412.15115.

[20] Traian Rebedea et al. NeMo guardrails: A toolkit for controllable and safe LLM applications with programmable rails, 2023. arXiv:2310.10501.

[21] Sander Schulhoff et al. Ignore this title and HackAPrompt: Exposing systemic vulnerabilities of LLMs through a global scale prompt hacking competition, 2023. arXiv:2311.16119.

[22] Sam Toyer et al. Tensor trust: Interpretable prompt injection attacks from an online game, 2023. arXiv:2311.01011.

[23] Eric Wallace et al. The instruction hierarchy: Training LLMs to prioritize privileged instructions, 2024. arXiv:2404.13208.

[24] Edwin B. Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927.

[25] Jingwei Yi et al. Benchmarking and defending against indirect prompt injection attacks on large language models, 2023. arXiv:2312.14197.

[26] Qiusi Zhan et al. InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents, 2024. arXiv:2403.02691.

[27] Lianmin Zheng et al. Judging LLM-as-a-judge with MT-bench and chatbot arena, 2023. arXiv:2306.05685.