# Full Experiment Review – Critical Evaluation

## Introduction & Experiment Overview

This project presents a **7-phase evaluation** of defenses against prompt injection attacks, progressing from measuring baseline vulnerabilities to developing detectors, optimizing them, and finally testing on challenging attack scenarios[[1]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/README_FINAL.md#L39-L47). The approach is systematic and logical:

* **Phase 1** establishes the severity of prompt injection (attacks on LLMs).
* **Phases 2–5** focus on building an input-side detection system (signature-based, rule-based, and learned fusion detectors, plus an obfuscation normalizer) and optimizing it for high true positive detection with zero false positives.
* **Phases 6a–6c** validate the system on **(6a)** benign inputs with obfuscation (to ensure no false alarms), **(6b)** *novel attacks* not seen in training, and **(6c)** *adversarial attacks* crafted to evade detection.

Throughout, the experiment reports detailed metrics (TPR, FPR, F1, etc.) with confidence intervals and statistical tests, and it openly discusses limitations. As a critical reviewer, I will examine each phase’s methodology and claims, checking for sound logic, credible numbers, and properly scoped conclusions. Overall, the multi-phase structure provides a cohesive narrative: the defense evolves in response to identified weaknesses, and the final evaluations ensure the claims are not overstated.

## Phase 1: Baseline Vulnerability Assessment

**Method:** The team first measured how vulnerable state-of-the-art models are to prompt injection. They used *400 queries* (200 benign + 200 attack attempts across 8 evasion techniques) and tested models including **LLaMA-2 7B** and **Falcon-7B**[[2]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase1/PHASE1_COMPLETION_SUMMARY.md#L39-L44). Attack Success Rate (ASR) was defined as the percentage of malicious prompts that caused the model to produce disallowed or compromised output.

**Results:** The attacks proved **highly effective on LLaMA-2** – *65% ASR* (130/200 attacks succeeded, 95% CI ≈ [55%, 74%])[[2]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase1/PHASE1_COMPLETION_SUMMARY.md#L39-L44). By contrast, **Falcon-7B** was far more resistant (*~5% ASR*, only ~10/200 succeeded)[[2]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase1/PHASE1_COMPLETION_SUMMARY.md#L39-L44). This stark difference suggests that vulnerability is model-dependent (perhaps due to different alignment or architecture). Within the attack set, some techniques were universally potent: e.g., *“plain” prompt injections and delimiter or role-confusion attacks had 100% success on LLaMA-2*[[3]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase1/PHASE1_COMPLETION_SUMMARY.md#L40-L44). Others were less effective – notably a **homoglyph attack** (replacing characters with look-alikes) achieved 0% success on both models[[3]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase1/PHASE1_COMPLETION_SUMMARY.md#L40-L44), indicating that certain simple obfuscations didn’t fool the models’ built-in filters.

**Soundness:** The Phase 1 methodology is straightforward and well-documented. They used a sufficiently large sample of attempts and even computed Wilson confidence intervals, which lends credibility by quantifying uncertainty (e.g. LLaMA-2’s 65% ± ~10% CI)[[2]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase1/PHASE1_COMPLETION_SUMMARY.md#L39-L44). The extreme difference between models (65% vs 5%) might raise a reviewer’s eyebrow – is it the model or the setup? The report hints these were *“RAG-borne”* attacks[[4]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/README_FINAL.md#L8-L11), implying the attacks might involve retrieval-augmented generation context. If so, perhaps Falcon’s retrieval or refusal system was stricter. This would be worth a brief explanation in a publication, but it doesn’t invalidate results – if anything it shows the range of possible behaviors. The key takeaway, which the authors state, is that **prompt injections can be highly effective** (at least on LLaMA-2), establishing the need for robust defenses. The logic here is sound and not exaggerated – claiming “attacks are highly effective” is justified by the data[[4]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/README_FINAL.md#L8-L11).

One minor critique: The summary in Phase 1 lists *“attacks blocked: 130 (32.5%)”* and *“attacks succeeded: 70 (17.5%)”*[[5]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase1/PHASE1_COMPLETION_SUMMARY.md#L134-L142), which appears to be calculated over all 400 queries (including benign). This is a bit confusing since ASR is usually reported out of attacks only. It might be clearer to state success rates out of the 200 attacks for each model separately (as they did with 65% and 5%). This is a presentation detail – not an error, but something to clarify for readers. Overall, Phase 1 provides a credible baseline that justifies the defense strategy pursued next.

## Phase 2: Developing Input-Side Detectors

Having shown the threat, the project pivots to detection. **Phase 2** implements **input-side detection**, meaning the system scans user prompts (and retrieved context) *before* they reach the LLM[[6]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L12-L20). This is a crucial design choice: a failed earlier attempt at *response-side* detection only caught ~1.5% of attacks (essentially useless, since it’s *after* the model has already produced a possibly harmful response)[[7]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L16-L19). Input-side detection is proactive.

**Detectors:** Three lightweight detectors were built (no heavy ML initially):

* **v1: Signature-based.** A set of known malicious phrases and regex patterns (e.g., "ignore all previous instructions", "pretend you are...", suspicious delimiters like ---, Base64 strings, etc.)[[8]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L35-L44)[[9]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L46-L54). This flags inputs containing these telltale markers. It’s essentially a keyword blacklist with a bit of fuzziness. Complexity ~100 lines, sub-millisecond runtime[[9]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L46-L54).
* **v2: Heuristic rules.** Builds on v1 but adds heuristic anomalies: e.g. high symbol density, long runs of numbers, odd Unicode symbols, YAML/JSON injection patterns, excessive uppercase, etc.[[10]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L52-L60). These rules catch inputs that *look strange* even if they don’t hit a known keyword. Complexity ~200 lines, ~2ms runtime[[10]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L52-L60)[[11]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L62-L70).
* **v3: Statistical anomaly detection.** This further adds simple stats like Shannon entropy of characters, distribution of word lengths, presence of repeated patterns, etc.[[12]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L65-L73). The idea is to flag inputs that are statistically unlike normal user queries (e.g., high entropy from random encoding or too many weird characters). Complexity ~300 lines, ~3ms[[13]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L64-L72).

All detectors were evaluated on the Phase 1 dataset (the 400 queries, where 70 were actual successful attacks needing detection). The performance was **surprisingly strong**:

* **True Positive Rate (TPR):** ~78–81% across v1, v2, v3[[14]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L83-L91). Concretely, v1 caught 55 of the 70 successful attacks (78.6%), and v2/v3 caught 57/70 (81.4%)[[15]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L95-L103). In other words, even the simple keyword filter v1 detected the majority of attacks. The added rules and stats in v2/v3 *only improved detection by 2 attacks* (~2.8% gain)[[16]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L124-L132).
* **False Alarm Rate (FAR):** 0% on the 200 benign queries for all three detectors[[17]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L85-L91). This is excellent – none of the benign questions (like *“What is machine learning?”*) were mistakenly flagged. The 95% CI for FAR was [0%, ~1.9%] given 0/200 false positives[[18]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L86-L89), so we can be fairly confident the true FAR is very low. Precision was correspondingly 100% (no false positives, so every alert was indeed an attack)[[19]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L114-L122).

The **critical insight** here is that *input-side scanning works* – it achieved ~80% TPR whereas earlier output-based methods failed. The detectors introduced effectively *no user friction* (zero false alarms)[[19]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L114-L122). The numbers are credible and actually realistic: ~80% coverage of known attacks means some attacks still bypass single detectors, which is expected. The report explicitly notes that these ~80% TPR figures are *“not artificially inflated by success tokens”* (i.e., they’re measuring actual attack content, not relying on an oracle)[[20]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L109-L117).

**Statistical independence & utility of v2/v3:** The team conducted McNemar’s tests to see if the small differences between detectors were significant. **They found no statistically significant improvement** from v2 or v3 over v1 alone on this dataset (p ≈ 0.16)[[16]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L124-L132)[[21]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L134-L143). In practical terms, *v1’s simple signature matching was doing nearly as well as the more complex detectors for these known attack types*. This is important: it suggests that many Phase 1 attacks contained obvious phrases that v1 could catch, and the exotic rules/entropy didn’t add much for *that particular set*.

As a critical reviewer, I find Phase 2 solid. The detectors are described clearly and seem appropriate for the threat model. The results (80% detection, 0% false positive) are believable given the methodology – they likely overfit to known attack patterns, but Phase 2 was about addressing known attacks. The authors properly highlight that v1 covers most cases and additional complexity had diminishing returns[[16]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L124-L132). This honesty is good; it tempers any impulse to claim v2/v3 are “must-haves” if data doesn’t support it. The lack of immediate gain from v3 foreshadows that a more powerful approach might be needed later (which indeed comes in Phase 5).

**Suggestion:** One thing to double-check is how these detectors handle *attack attempts that ultimately failed* in Phase 1. The Phase 2 evaluation focused on the 70 successful attacks, but what about the other 130 attempts that *the model itself blocked*? In deployment, those are still malicious inputs that a detector should ideally flag (even if the model didn’t comply). If those were included, v1’s overall TPR on all attack *attempts* might be lower or higher. The final report’s summary indicates v1 flagged ~80% of *all* attacks attempts (not just successful)[[22]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/README_FINAL.md#L132-L140). For instance, v1 had 80% TPR and v3 57% on Phase1 attacks overall[[22]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/README_FINAL.md#L132-L140). This discrepancy (57% vs 81% in different places) initially confused me, but it seems Phase 2’s 81.4% is on the subset of successful attacks, whereas 57% was likely v3’s coverage over *all 200 attempts*. The authors might clarify this in a camera-ready version to avoid confusion. Nonetheless, the qualitative finding stands: signature detection is effective but can be complemented by other methods for broader coverage.

## Phase 3: Optimizing Detector Combinations

Phase 3 explores whether combining detectors can improve coverage without introducing false positives. Since v1, v2, v3 each caught slightly different attacks, an ensemble might do better. They evaluated **7 configurations**: all combinations of {v1, v2, v3} (including each alone)[[23]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L12-L20). The detection logic for combos was essentially “OR fusion” – flag an input if *any* detector flags it. (At this stage, no weighted scoring or ML fusion yet, just binary OR.) All configs maintained 0% FAR (since each detector individually had 0 FAR on known benign, any OR of them also had 0 FAR). So the comparison is purely on TPR (detection rate).

**Results:** The combined results had some surprises:

* The **best-performing combo** was reported as **v1 + v2**, catching 81.4% of attacks (57/70)[[24]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L16-L24). This is slightly higher than v1+v3 (which was 78.6%, 55/70)[[24]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L16-L24). Adding all three (v1+v2+v3) also gave 81.4%, no further gain over v1+v2[[24]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L16-L24). Essentially, v2 and v3 were overlapping in what they caught (both found the 2 extra attacks v1 missed, and v3 alone by this point was ineffective – more on that below). The table shows v1+v2 and v1+v2+v3 had identical performance[[25]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L20-L22)[[26]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L34-L40).
* Interestingly, **v3 (classifier) alone showed 0% TPR in this phase’s table**[[27]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L16-L20)[[26]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L34-L40). This indicates that whatever “v3” was at this point, it did *not* catch any of the 70 attacks by itself. In Phase 2, v3 was a statistical detector that did catch 57/70, so why 0% now? It turns out there was a correction: the document is labeled *“(CORRECTED – Independent Detectors)”*, implying an earlier mistake treating detectors as independent when they weren’t[[28]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L1-L8). Possibly the initial v3 used Phase 1 knowledge or required a threshold that wasn’t properly set. My interpretation: **v3 was likely intended to be a trained classifier (e.g., using an ML model or OpenAI’s moderation API)**, and in a truly independent test it failed to detect any attacks. In other words, a generic content classifier didn’t recognize prompt injection attempts as malicious (not surprising – “Ignore previous instructions” isn’t toxic or explicit content, so a moderation filter would pass it). This *0% TPR for classifier-only* underscores why custom detectors (v1, v2) are needed; off-the-shelf classifiers don’t flag these attacks[[26]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L34-L40). The authors do not hide this result – they include the 0% in the table, which adds to credibility. They label v3 as “Classifier-only” in the table[[26]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L34-L40), suggesting it might have been an experiment with an ML model that was not trained properly on this task.

**Pareto Analysis:** Given all combos had 0 FPs, the team looked at complexity vs TPR. Config **D (v1+v2)** was chosen as **Pareto-optimal**: it had the highest TPR (81.4%) among 2-detector combos[[29]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L14-L22)[[30]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L24-L32), and using 2 detectors is simpler than using all 3 for the same effect. They noted that config E (v1+“classifier”) and G (v1+v2+v3) were statistically equivalent to D in performance[[31]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L50-L59), but D uses fewer components[[32]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L61-L69). So they recommended **v1+v2 as the primary defense** at this stage. Config D caught 57 of 70 attacks, missing 13 (TPR ~81%)[[33]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L138-L141). The *marginal* benefit of adding v2 was that v1 alone caught 55, v2 alone 30, and v2 contributed 2 unique catches that v1 missed[[34]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L126-L134)[[33]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L138-L141). Those likely involved statistical anomalies not covered by v1’s signatures.

**Critical view:** Phase 3’s analysis is thorough, even including a statistical significance check. They confirmed v1+v2’s advantage over v1 was not significant at p=0.157 (only +2 detections)[[35]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L50-L58), but since v1+v2 had the highest point TPR and essentially no downsides, it was a reasonable choice. The logic that *fewer components = less complexity/latency* is valid; v1+v2 being chosen over v1+v3 at this point seems to hinge on the assumption that the “classifier/v3” wasn’t adding value. As a reviewer, I note a slight inconsistency: earlier, v3 (statistical) was as good as v2; but here “classifier” (likely meaning a different v3) was worthless. This discrepancy might confuse readers – the authors should clarify that the v3 in Phase 3 was not the same implementation as in Phase 2 (perhaps they attempted a ML classifier and it didn’t pan out). They do mention “independent detectors” and label v3 as classifier, which hints at this difference[[28]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L1-L8)[[26]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L34-L40). For publication, explaining that an out-of-the-box classifier was tested and failed would be useful context, reinforcing why the custom approach was needed.

That aside, the outcome (v1+v2 best) is plausible given many attacks had obvious cues (v1) and a few had only odd patterns (caught by v2). The team’s decision to prioritize the simpler, proven combo is prudent. They did not over-claim anything here – in fact, they downplay v3’s usefulness given the data. This builds trust: they are clearly letting the results drive conclusions rather than an attachment to any detector.

## Phase 4: Threshold Robustness Analysis

Phase 4 addresses a potential concern with detection systems: sensitivity to threshold tuning. Detectors like v2/v3 might produce a score or require a cutoff. They took the best config from Phase 3 (v1 + v3 **OR-fusion** – interestingly they used v1+v3, not v1+v2, perhaps because by now they refined v3 or wanted to include a classifier score) and swept the decision threshold from 0.05 to 0.75[[36]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/PHASE4_COMPLETE_SUMMARY.md#L4-L12)[[37]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/PHASE4_COMPLETE_SUMMARY.md#L14-L22).

**Key finding:** *“Threshold-invariant performance”* – astonishingly, **all thresholds in [0.05, 0.75] yielded the exact same TPR (87.0%) and FAR (0.0%)**[[37]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/PHASE4_COMPLETE_SUMMARY.md#L14-L22). In other words, there is a **flat region where any reasonable threshold works**, with the system catching 87% of attacks and no false alarms regardless. This implies the positive and negative samples were **cleanly separated** in whatever scoring space they used[[38]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/PHASE4_COMPLETE_SUMMARY.md#L16-L24). Likely, the OR-fusion made this trivial: if v1 or v3 fires, it’s an attack (score=1), else 0 – so unless the threshold is above 1 or below 0, it’s the same. The report indeed notes that this demonstrates *“excellent discrimination... making threshold selection non-critical”*[[38]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/PHASE4_COMPLETE_SUMMARY.md#L16-L24). They recommended simply using 0.50 for convenience[[39]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/PHASE4_COMPLETE_SUMMARY.md#L28-L36)[[40]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/PHASE4_COMPLETE_SUMMARY.md#L30-L38).

From a critical standpoint, this result is a bit *too* perfect, but given the nature of OR-fusion, it’s actually plausible. Essentially, v1+v3 (as of Phase 4) was likely implemented such that an input gets a score of 1 if flagged by the detectors, 0 otherwise. The threshold sweep confirms there’s a wide range where outcomes don’t change (as long as threshold is between 0 and 1, any attack yields score=1 which is above threshold, any benign yields 0 which is below). So the “sweep” was perhaps a sanity check. It’s reported with enthusiasm (they call it a *“remarkable finding”*[[41]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/PHASE4_COMPLETE_SUMMARY.md#L12-L19)), which in a paper might be toned down because it’s somewhat self-evident given binary detectors. However, they might have had in mind a scenario where v3 produced a continuous anomaly score – the invariance suggests maybe all anomaly scores for attacks were above all benign scores across that range. If that’s the case, it *is* a notable robustness feature (no finely tuned threshold needed).

Either way, Phase 4’s conclusion that threshold tuning is “not critical” is supported by their data[[38]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/PHASE4_COMPLETE_SUMMARY.md#L16-L24). They tie this to operational benefits: simpler deployment, no risk of performance drift if the threshold isn’t perfect[[42]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/PHASE4_COMPLETE_SUMMARY.md#L26-L34). They even position it as a **differentiator** (most ML systems do require careful threshold tuning; theirs apparently doesn’t)[[43]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/PHASE4_COMPLETE_SUMMARY.md#L42-L47). A skeptical reader might think: “Is the system really that robust, or was the evaluation too limited (only known attacks and benign) making it easy to separate?” The answer likely lies in Phase 6a: when we introduce obfuscation in benign queries, thresholds might matter more (since some benign got scores near the cutoff). But under the controlled Phase 1 data, indeed there was a clear margin.

The team’s interpretation is fair – they don’t claim this will hold for all time, just that within the evaluated range it did. They also cite a *flat ROC curve (TPR vs FPR)* as a visual (likely a horizontal line at 87% TPR until FPR jumps from 0 to something once threshold goes below 0.05 or above 0.75)[[44]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/PHASE4_COMPLETE_SUMMARY.md#L88-L96). All of this indicates the system at this point was **operationally stable**. No concerns here; Phase 4 is a minor but nice point reinforcing that their chosen detection signals separate known attacks from benign very cleanly.

*(Side note: The config used was “v1 + v3 (Signature + Classifier)”*[*[36]*](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/PHASE4_COMPLETE_SUMMARY.md#L4-L12)*. Given earlier Phase 3 preferred v1+v2, I suspect by Phase 4, “v3” refers to an improved detector possibly replacing v2. It could be that by now they treated v3 as the anomaly detector and dropped v2 entirely. This isn’t explicitly explained, but since they carry v1+v3 forward, I infer they realized v3’s feature-based approach might generalize better than v2’s specific rules. This doesn’t undermine any results, but an explanation in text would be helpful to avoid confusion between phases.)*

## Phase 5: Learned Fusion & Obfuscation Defense (Nested CV)

Up to Phase 4, the system was essentially rule-based. Phase 5 is a major enhancement: they introduce a **Normalizer** to handle complex obfuscation and a **learned fusion model** (logistic regression) to combine multiple signals optimally[[45]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/README_FINAL.md#L92-L101)[[46]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/README_FINAL.md#L109-L118). The normalizer performs text preprocessing: Unicode NFKC normalization, stripping zero-width chars, mapping homoglyphs to their ASCII equivalents, detecting mixed-script inputs, etc.[[45]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/README_FINAL.md#L92-L101). This should nullify attacks that rely on character tricks (and also ensure benign queries with such characters aren’t mistakenly flagged). The logistic regression (“fusion”) takes as features: the outputs of v1, v2, v3 detectors (yes, they appear to use all three detector signals internally), plus normalizer metrics (counts of weird chars, etc.), plus some text stats[[47]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L55-L63). In total 25 features including one-hot encodings of which rule fired, text length stats, etc.[[47]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L55-L63). The logistic model then outputs a probability of “attack” vs “benign”. This is a more nuanced decision mechanism than the simple OR, potentially allowing it to catch subtle patterns (and also potentially reduce false positives by weighing signals).

**Validation method:** They used **nested cross-validation** to train and evaluate this fusion model robustly[[48]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L30-L39). This is excellent practice – it prevents overfitting and ensures the threshold for decision is chosen purely on training data in each fold. Specifically, they did 5-fold outer CV for evaluation, and within each training split, a 3-fold inner CV to sweep the threshold and choose the highest threshold that yields 0% FPR[[49]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L36-L44). The objective was to find an operating point with **zero false positives** (primary requirement) and as high TPR as possible. This nested CV avoids “threshold leakage” (using test data to set the threshold), which a critical reviewer appreciates[[50]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L146-L152).

**Results:** The fusion model *dramatically outperformed* the earlier baseline:

* **TPR ~99.0%** with 0.0% FPR at the selected operating point[[51]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L13-L21). Specifically, it caught 198 out of 200 attacks (across all folds) while still not flagging any of 200 benign[[52]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L77-L85). This is a **+12 percentage point improvement** over the ~87% TPR baseline (v1+v3 OR)[[53]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L116-L124). In real terms, it means 24 more attacks are detected that previously slipped through[[53]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L116-L124). That is substantial – essentially only 2 out of 200 attacks got by (compared to 26 missed before)[[54]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L118-L124). Precision remained 100% (no false alarms)[[53]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L116-L124).
* They report an F1 score of ~0.995, up from 0.93[[53]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L116-L124). With zero FPs, F1 is mostly driven by recall, so that checks out given ~99% recall.
* The 95% CI for TPR was [95.0%, 100.0%][[51]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L13-L21), reflecting uncertainty but clearly showing a significant gain over the 87% level. The FPR CI is [0%, 0%] (since no FP were observed)[[51]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L13-L21).

This is an impressive result, perhaps *too* impressive at first glance – how did they achieve near-perfect detection? The likely reason: by training a model on the known attack patterns, the logistic regression learned to recognize combinations of features that indicate an attack, even for those few attacks that v1+v2 had missed. For example, perhaps some attacks that v1 missed had subtle cues that v2/v3 features picked up when combined (like high entropy plus a suspicious JSON structure might indicate an attempted injection that individually didn’t trigger a rule). The fusion essentially “learned” these last few patterns.

**Crucially**, because they did proper cross-validation, this 99% isn’t just overfitting one dataset – it’s validated on held-out folds. This means if we ran the system on new data with similar attack styles, we’d expect ~95–100% detection of those known types. They even highlight that their approach **exceeds a simpler baseline**: they compare to a “Signature+Classifier at t=0.5” baseline (which had 87% TPR)[[55]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L19-L22)[[56]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L98-L106). The fusion gave +12 points TPR[[53]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L116-L124). (This baseline presumably was the Phase 3 config E or Phase 4 method – basically the best earlier approach.)

From a **critical perspective**, Phase 5 demonstrates strong rigor. They did nested CV, which is great. They clearly report the lift over baseline[[53]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L116-L124). They even discuss alternative operating modes: if one were willing to tolerate some false alarms, one could lower the threshold and get 100% TPR at ~12% FAR (they mention a “High-Recall Mode: ~100% TPR with ~12% FPR” as an option for offline monitoring)[[57]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L138-L143). This shows they are not hiding trade-offs; they found the sweet spot for production (zero-FP) but acknowledge you *can* trade precision for recall if desired. That balanced view is exactly what a critical reviewer wants to see.

One potential concern: The fusion model’s features include the outputs of v1, v2, v3 detectors[[47]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L55-L63). Since those already achieved near-perfect precision and high recall, the logistic is heavily relying on engineered features that were tailored to the Phase 1 attacks. In other words, *the fusion is almost certainly overfit to known attack patterns* (despite cross-val mitigating overfitting to specific samples, the model still may not generalize to new attack strategies). The authors appear aware of this, as the next phases test generalization. As a reviewer, I’d caution that *99% TPR on the test set that is very similar to training conditions is great, but the real question is how it performs on genuinely novel attacks or slight variations.* The team addresses exactly that in Phase 6b/6c.

Another point: The normalizer was integrated here, but Phase 5’s results (99% TPR, 0% FPR) were mostly on the original data which had *some* obfuscated attacks (e.g., homoglyph, ZWJ attacks were part of Phase 1’s 8 types). Achieving 99% TPR suggests the normalizer + features helped catch even those homoglyph attacks that v1 or v3 might have missed. In Phase 1, “homoglyph” attacks had 0% ASR (they didn’t succeed on the LLM)[[58]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase1/PHASE1_COMPLETION_SUMMARY.md#L42-L44), so they might not have been in the 70 “successful” attacks at all. But if any obfuscation attack was in the set of attempts, the normalizer would ensure detection flags it easily. So the fusion system likely had no trouble with things like ZWJ or homoglyph – a point proven in Phase 6a next.

In summary, Phase 5’s methodology is strong and the claims (99% detection of known attacks with zero false alarms) are backed by thorough validation[[59]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L76-L84)[[56]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L98-L106). There is no sign of overconfidence: they present this as a **primary operating point** for production[[60]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L11-L19), which is justified for attacks similar to those seen. They don’t claim “we solved prompt injection” – instead, they frame it as *deploy this fused model as the primary defense, as it outperforms the earlier signature-only approach*[[61]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L19-L25). This is appropriate. A critical reviewer might only add: ensure to communicate that these results apply to attacks drawn from the same distribution as training (Phase 1 types). The authors do exactly that by introducing novel attack tests in the next phase.

*(On a publication readiness note: the Phase 5 report is detailed with tables per fold, etc. For a paper, one might not show all that, but the key summary – 99%±2.2% TPR @ 0% FPR vs baseline 87% – would be a headline figure*[*[51]*](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L13-L21)[*[53]*](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L116-L124)*. The inclusion of confidence intervals and proper cross-val is a sign of a polished, rigorous study.)*

## Phase 6a: Benign Obfuscated Input (False Alarm Check)

Before celebrating the Phase 5 model, the team wisely conducted **Phase 6a** to test a critical aspect: Does the system accidentally flag *benign* queries that contain unusual text? In practice, users might use emojis, foreign languages, or formatting that the detectors haven’t seen. A robust defense should not over-react to these. Phase 6a created **260 benign samples** with various **obfuscation patterns**[[62]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L28-L37), grouped into categories: Clean (no obfuscation), Unicode/Emoji, Homoglyphs (e.g. replacing letters with Cyrillic look-alikes), Zero-Width characters inserted, Mixed-Script (legitimate bilingual queries), Odd Formatting (excess spaces, newlines, case mixing), Symbol-heavy, and Non-Latin scripts (Russian, Chinese, Arabic text that is benign)[[63]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L30-L39). Importantly, *all 260 are benign questions*, just distorted. This directly tests false positive rates under worst-case benign conditions.

**Results:** The system’s performance here underscores why the **Normalizer** was added:

* The recommended production config **Normalizer + v3** (i.e. use the normalizer then the semantic/logistic detector without the brittle v1) achieved **FAR = 0.77%** (only 2 false positives out of 260)[[64]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L13-L19). This is **below the 1% false alarm goal** set in Phase 5[[65]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L200-L208). Essentially, it means 99.23% of benign queries, even with heavy obfuscation, passed through correctly. Both false positives came from minor cases: one in the “Formatting” category and one in “Symbol” category (each 3.3% FAR in that category)[[66]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L80-L88). For example, an extremely oddly formatted benign query might have triggered a rule threshold. But zero false alarms were observed for Homoglyph, Mixed-script, Non-Latin text, etc. – all those were handled perfectly[[66]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L80-L88).
* By contrast, **v1 (signature) alone was disastrous**: 23.1% FAR overall (60/260 benign flagged)[[67]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L15-L18)[[68]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L57-L65). In particular, v1 *falsely flagged 100% of benign queries with homoglyph characters* (e.g., “machіne learning” where the 'i' is Cyrillic)[[69]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L68-L76). This happened because v1 likely has a regex for known bad phrases and, without normalization, the Cyrillic 'i' is not recognized as such, possibly messing up pattern matching or triggering some “uncommon character” rule. Similarly, v1 erroneously flagged many mixed-script and symbol-laden queries[[69]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L68-L76). This confirms the concern that signature rules can misfire on benign inputs that look unusual.
* **Normalizer+v1 improved that hugely** (homoglyph benign went from 100% false alarm to 0%)[[66]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L80-L88)[[70]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L90-L98), but **still had 11.5% FAR overall**[[71]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L61-L64). The normalizer fixes the character issues (homoglyph, zero-width) – which it did: homoglyph benign FAR dropped to 0% after normalization[[66]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L80-L88). But v1 then still flagged other things like mixed-script content (because the normalizer intentionally leaves true mixed-language text as-is to not corrupt meaning – there’s a safeguard)[[72]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L120-L128). v1’s rules, even with clean text, don’t cope well with, say, a query partly in Russian. So 11.5% FAR is still too high for production[[73]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L144-L153).
* **v3 (semantic/logistic) alone** had 0.77% FAR (same 2/260 mistakes as above)[[67]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L15-L18)[[74]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L59-L64). This shows that the learned detector by itself is robust to these benign variations – likely because it doesn’t rely on brittle keyword matching. The two mistakes likely occurred because those particular benign inputs tripped some feature threshold (e.g., *one benign query had an extremely odd layout that the model found similar to an attack’s format by coincidence*). But 2 errors out of 260 is very low, and importantly none of those errors were in critical categories like foreign language or homoglyphs – they were minor formatting cases[[75]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L84-L92).
* **Normalizer + v3** was the same 0.77% FAR[[74]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L59-L64). In theory, normalizer shouldn’t change benign queries much except remove invisible chars, etc. Indeed, the normalizer mostly made a difference for v1; v3 was already good. But combining them ensures even edge cases like ZWJ or homoglyph are cleaned (which might have helped avoid *potential* v3 mistakes, though v3 had none in those categories anyway)[[66]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L80-L88).
* **v1+v3 (without normalizer)** had a high FAR (23.8%)[[74]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L59-L64), even worse than v1 alone in this test. This is because *OR-ing v1 with v3 means any false positive from v1 will trigger an alert* even if v3 is fine[[76]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L156-L165). So v1’s propensity to flag weird benign queries drags the combo down – a classic precision vs recall trade-off. This confirms that **including the signature detector v1 in the final ensemble is problematic for production**. It catches a few more attacks, but at the cost of many false alarms on innocuous inputs[[76]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L156-L165).
* **Normalizer + v1 + v3** lowered that FAR to 12.3%[[64]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L13-L19)[[74]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L59-L64) (normalizer fixed the worst v1 issues, halving the false alarms, but 12% is still high)[[77]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L13-L21). The authors correctly mark this as *“needs tuning”* and not production-ready[[78]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L19-L22).

**Analysis:** Phase 6a provides strong evidence that the system is *safe to deploy* if configured properly. The choice of **Normalizer + v3 as the primary defense is validated** – it meets the ≤1% FAR goal on tough benign inputs[[65]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L200-L208)[[79]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L220-L228). The data justifies the recommendation to drop v1 in production (to avoid its false positives) and only use the full Normalizer+v1+v3 in a high-recall *monitoring mode* where some false alarms are acceptable[[80]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L180-L188). This is a very important nuance and they handle it well, giving separate recommendations for production vs monitoring[[81]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/README_FINAL.md#L141-L149).

From a critical perspective, the Phase 6a experiment design is excellent. They used 20 base benign queries and systematically generated variations in 8 obfuscation categories[[82]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L40-L48). This is comprehensive and covers likely sources of false positives. The sample size (260) is decent for a lab test, though a real deployment would face many more benign queries – but since 0 false positives were seen in most categories, it’s reassuring. They also clearly list limitations: e.g., these are synthetic obfuscations, only 20 base queries, primarily English text with some non-English, and they did not include *adversarial* obfuscations (someone actively trying to fool the normalizer)[[83]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L210-L218). A critical reviewer would concur – those are reasonable limitations. In particular, *adversarial obfuscation* (like deliberately designing a homoglyph mapping the normalizer doesn’t catch, or a ZWJ pattern that slips through) is not tested here. The authors flag that themselves as future work[[83]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L210-L218). This transparent acknowledgment prevents overconfidence in the normalizer: it works on standard obfuscations, but a clever attacker could invent new ones (Phase 6c will cover some aspect of this).

One more observation: The success of the normalizer on benign input (0% FAR on homoglyph, mixed, non-Latin) is a strong validation of its design[[66]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L80-L88)[[70]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L90-L98). For instance, v1 had 100% FAR on homoglyph benign queries, which normalizer reduced to 0% by mapping those characters to ASCII[[69]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L68-L76)[[66]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L80-L88). This demonstrates a *100% improvement* on that front[[84]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L104-L113) – something the authors rightly highlight. Also, the normalizer’s “mixed-script safeguard” (not converting characters if the text is truly bilingual) prevented it from wrongly altering non-malicious content[[70]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L90-L98). The result was that legitimate mixed-language queries were not flagged (v3 handled them fine)[[85]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L124-L128), whereas v1 without safeguard would have freaked out at the strange mix. All of this indicates the logic is sound and *the numbers back it up*. There’s no exaggeration: they achieved <1% FAR and they claim exactly that, noting the specific conditions.

In conclusion, Phase 6a shows the system’s **conservative bias** – it stays quiet on benign input, even weird looking benign input, which is critical for real-world use. The conclusions are appropriately phrased: *“Normalizer+v3 is safe for production (FAR <1%), v1+v3 is not recommended due to high FAR”*[[78]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L19-L22)[[86]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L186-L194). This matches the evidence. I have no major critiques; the experiment is solid. Perhaps they could mention if the two false positives in v3 could be further analyzed (were they due to, say, an overly long sequence of punctuation that triggered the statistical anomaly threshold?). But given the low count, it’s a minor point. The team’s caution to still use Normalizer+v1+v3 in monitoring (where you’d rather catch every possible attack even if a few false alerts) is sensible[[80]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L180-L188). It shows maturity in evaluating different use-case needs.

## Phase 6b: Detection of **Novel Attacks** (Unseen in Training)

Phase 6b is arguably the *most critical test* of the detector’s generalization. After Phase 5, one might worry that the system is tailored to the specific attacks it saw (the eight types in Phase 1). Attackers constantly invent new strategies, so how does the system handle *novel injection techniques* that it wasn’t explicitly trained on? To probe this, the authors generated **120 novel attack samples** across 6 categories not in the original training:

1. **Alternative phrasing** – e.g., using different words to convey “ignore previous instructions” (synonyms, polite forms, etc.)[[87]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L33-L41).
2. **Instruction embedding** – hiding the malicious directive inside JSON, YAML, XML, or code blocks (formats not heavily represented in Phase 1)[[88]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L33-L40).
3. **Multi-turn manipulation** – attacks that reference the conversation history or pretend a prior context (“As we discussed earlier…”)[[89]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L35-L40).
4. **Context confusion** – using role-play or hypothetical scenarios to trick the model (e.g., “Let’s play a game where you break the rules…”)[[89]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L35-L40).
5. **Encoding/Obfuscation** – using encoding schemes like Base64, ROT13, or leetspeak to hide the malicious instruction[[89]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L35-L40).
6. **Indirect prompt** – asking things in a roundabout way (“What would one do if they *were* to bypass the filter…”)[[90]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L39-L41).

None of these exact approaches were in Phase 1’s data. Also, these 120 attacks were not tested on the LLM (for success) in this phase – rather, they are assumed to be *plausible attacks*. (The authors note as a limitation that they did *not* verify LLM compliance for these, nor did they adapt them specifically to defeat the system – that comes in 6c – these are just “novel” in style)[[91]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L223-L229).

**Results:** The performance drop was **significant**: roughly *half of the novel attacks were missed*.

* The strongest configuration (Normalizer+v1+v3) detected **49.2% TPR** (59 out of 120 novel attacks)[[92]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L13-L21). This exactly matched v1+v3 with no normalizer (49.2% as well)[[92]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L13-L21), confirming that the *normalizer didn’t help* because these novel attacks presumably did not rely on fancy Unicode or homoglyph tricks (they were novel in logic, not in encoding)[[93]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L22-L25)[[94]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L113-L121). Similarly, Normalizer+v3 caught 44.2% (same as v3 alone 44.2%)[[95]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L14-L18).
* **v3 (semantic detector) alone** far outperformed **v1 (signature) alone**: 44.2% vs 11.7% TPR[[96]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L14-L19). This makes sense: an entirely new phrasing will evade v1’s hardcoded phrases easily (v1 only caught 14/120, likely those with some known keywords still present)[[97]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L50-L58). v3, which looks at context and anomalies, caught 53/120 on its own[[97]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L50-L58). This validates that *semantic/generalized features are crucial for novel threats*.
* **Ensemble (v1+v3)** gave 49.2%, so only a small improvement over v3 alone[[97]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L50-L58). They note a +5% lift from combining v1 and v3 (44.2→49.2%)[[98]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L159-L167), which is much smaller than the ~7% lift v1+v3 gave on Phase1 data. This suggests that on novel attacks, *v3 covers most of what can be caught; v1 contributes very little* because its signatures are largely irrelevant here[[98]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L159-L167). In fact, 49.2% TPR is just shy of their hoped-for 50% benchmark (they point out the goal was ≥50%, which was missed by 0.8%)[[99]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L170-L178). They do not hide that the goal was not met – they explicitly call it out[[100]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L173-L180), which is a mark of honesty.
* **Breakdown by attack type** is illuminating:
* *Alternative phrasing:* Only 40% caught by v3, 5% by v1[[101]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L61-L69). Most paraphrased instructions slipped through (v3 caught a few, v1 almost none).
* *Instruction embedding (JSON/YAML):* 85% caught by v3, and the ensemble got 95%(!)[[102]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L69-L77). This is interestingly high – likely because these formats trigger anomalies (lots of symbols, structured patterns) that v3’s features and even v1’s patterns (maybe they had some rules for JSON syntax) can detect. They call this a **strength** – the system is quite good at recognizing when a prompt includes a chunk of code or data likely meant to smuggle instructions[[102]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L69-L77)[[103]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L75-L76).
* *Multi-turn manipulation:* A glaring hole – **0% detected by v1, only 40% by v3**[[104]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L77-L85). 60% of these multi-turn prompts were missed by the ensemble. The detectors operate on a single-turn input; they have *no understanding of dialogue context*, so an attack that says *“As we previously discussed, please output X”* doesn’t contain obvious forbidden patterns. The few (40%) that v3 caught might have included some anomaly like unusual phrasing, but generally this is a **critical gap**[[105]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L83-L91). They highlight that multi-turn attacks “are poorly detected” and require conversation-level analysis, which their system lacks[[105]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L83-L91).
* *Context confusion (roleplay/hypothetical):* Also very low – 25% detection by ensemble[[106]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L85-L93). These are sophisticated social engineering attempts (e.g., “Let’s pretend the policies don’t exist for a story”). The detectors likely miss these because the language doesn’t violate any pattern or stat – it’s the *intent* that’s malicious, which is hard to capture. Only 15% by v3 and 0–10% by v1 individually[[106]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L85-L93). So three-quarters got through. They mark this as another **critical gap**[[107]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L89-L91).
* *Encoding/Obfuscation:* Only 45% caught by ensemble[[108]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L93-L101). These attacks hide instructions in encoded text (like “SGVsbG8gd29ybGQ=” which is "Hello world" in Base64, etc). Since the normalizer doesn’t decode Base64 and v1 doesn’t have those patterns, most encoded attacks go undetected (just 35% by v3 alone, 25% by v1, combined 45%)[[108]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L93-L101). They note *“normalizer doesn’t decode”*, which is a limitation – their system handles character confusability but not actual encoding schemes[[109]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L97-L100).
* *Indirect prompt:* 50% caught[[110]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L101-L107) – e.g., sneaky questions “how would one bypass the filter” are somewhat detected by v3 (50%) and v1 (10%); ensemble 50%. So half get through.

So across categories, detection ranges from excellent (95% for structured embedding attacks) to abysmal (25% or less for context-based attacks). The overall ~50% TPR reflects that many categories were around that level or below.

**Interpretation:** The authors handle these results appropriately. They identify the **specific coverage gaps** clearly – multi-turn context, roleplay context, and encoding attacks were problematic[[111]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L142-L151)[[112]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L154-L162). They draw insights such as *“v1 is brittle on novel attacks (only 11.7% TPR) – paraphrasing breaks it completely”*[[101]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L61-L69), and *“v3 is more robust (44.2% TPR) but still misses over half”*[[113]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L132-L140). One insight: *“Semantic approach catches more novel attacks than signature matching”*[[114]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L134-L140) – this justifies why they invested in v3 and suggests future detectors should rely even more on semantic understanding (perhaps using an actual language model to judge intent).

They also note that combining v1 and v3 yielded only a small improvement, implying **v3 dominates** the ensemble’s capability on novel inputs[[98]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L159-L167). In fact, they observe that in Phase 1, v1 contributed a lot (80% vs 57%, big gap), whereas here v1 is nearly useless; thus ensemble lift is smaller[[98]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L159-L167). This indicates their Phase 5 fusion, which included v1, might not be very effective on new styles. It hints that maybe retraining or using a different approach for novel patterns is needed.

The **limitations** are explicitly listed: the novel attacks are *synthetic* and limited to 120 samples, and importantly they did *not* include adaptive attacks or test on actual model responses[[91]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L223-L229). As a reviewer I find this acceptable – Phase 6b’s goal was to test detection generality, not to measure end-to-end attack success. However, one might ask: *if some of these novel attacks wouldn’t actually cause the model to misbehave, is it fair to count them as misses?* The authors treat each novel attempt as something that *should* be caught if it were attempted. That’s a reasonable approach for a detector (better to be safe), but it means the detector is penalized even for not flagging an attack that might have failed anyway. They acknowledge they didn’t verify LLM compliance[[91]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L223-L229), so this is just a pure detection stress test. That’s fine.

**Recommendations:** They do not over-hype their system here – on the contrary, they emphasize that **51% of novel attacks evade detection**[[93]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L22-L25). The recommendation is to deploy the best config (Normalizer+v1+v3) but *“accept the gap”* and use monitoring to learn new attack patterns[[115]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L185-L193)[[116]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L192-L200). For future work, they propose concrete improvements targeting each gap: e.g., conversation history tracking for multi-turn, better NLU for context roleplay, adding a decoding layer for encoded attacks, expanding the signature list for alternative phrasing[[117]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L199-L207)[[118]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L210-L218). Each suggestion is tied to the observed weakness, which is exactly what a reviewer would want to see in the discussion. They also mention the effort level (high for multi-turn, medium for encoding)[[119]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L200-L209)[[118]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L210-L218), which is a nice touch to indicate feasibility.

The conclusions for Phase 6b are appropriately modest: the system that was great on known attacks is only **moderately effective (≈50% TPR)** on novel attacks[[120]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L232-L240). They call out the key achievement as identifying these gaps and proving that v3 is indeed more robust than v1 for novel cases[[121]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L234-L242). They recommend deploying with monitoring and planning improvements, rather than claiming the system handles everything[[122]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L236-L244). This is a balanced, honest assessment.

From a publication perspective, this is critical data. It shows the limits of the approach. The numbers and examples lend credibility – no silver bullet here, just transparency that more work is needed to handle evolving attacks. The fact that adversarial minds could come up with so many evasion tactics (and Phase 6b likely drew inspiration from known attacks in the wild or literature) means any static detector will have gaps. The authors confront this directly.

To nitpick, I might ask: How were those 120 attacks generated? By humans brainstorming, or using some automated prompt generation? It reads like they were manually designed per category (the descriptions in the table are very clear)[[87]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L33-L41). If manual, they likely covered obvious variants, but a real attacker might combine techniques or come up with others. The authors do say limited diversity (could expand to 500+ in future)[[91]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L223-L229), which shows they know 120 is not exhaustive. Perhaps including an example or two of each category in an appendix would strengthen it, but for the review, I think it’s fine.

## Phase 6c: **Adversarial (Adaptive) Attacks** Robustness

This final phase goes one step further than 6b: instead of just novel attacks, it considers **attacks explicitly designed to defeat the detection system**. In other words, a “red team” approach – assume the attacker knows roughly how your detector works (black-box or limited white-box knowledge) and crafts inputs to target those weaknesses. Phase 6c created **98 adversarial samples** in 5 categories[[123]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L31-L40):

* **Detector Evasion (20 samples):** Attacks carefully worded to *avoid* triggering v1 or v2 rules (e.g., no obvious keywords, perhaps very polite or indirect instructions).
* **Normalizer Evasion (19):** Attempts to break or confuse the normalizer – maybe using homoglyphs that aren’t in its mapping, or sequences that slip through its logic.
* **Threshold Evasion (20):** Inputs designed to produce borderline detector scores – perhaps by mixing benign content or reducing the intensity of malicious cues so that the ensemble’s confidence stays low.
* **Feature Confusion (19):** Inputs that exploit the logistic model’s features – e.g., lots of benign content with a small malicious part, or patterns that mimic the statistical properties of benign queries to fool entropy and other measures.
* **Gradient Attacks (20):** Likely generated by approximating the gradient of the classifier (the logistic regression) or using an attack algorithm (like TextFooler or an evolutionary approach) to minimize the detector’s output while keeping the attack semantics. These could include subtle word substitutions or phrasing tweaks aimed at the decision boundary.

These attacks are **not about succeeding on the LLM** per se, but purely about evading detection (some might overlap with Phase 6b’s novel styles, but the difference is the intent: Phase 6b’s were just new, Phase 6c’s are tailored to the known defense). They note that Phase 6c attacks target *“known system weaknesses”*, whereas Phase 6b’s novel attacks might inadvertently evade but weren’t specifically targeting the detector design[[124]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L39-L41).

**Results:** The numbers here are similar to Phase 6b, with overall detection around 50%, but the breakdown reveals the toughest challenges:

* **Overall TPR** for best config (Normalizer+v1+v3) was **53.1%** (52/98 caught)[[125]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L13-L21). Without normalizer (v1+v3) it was 54.1%[[126]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L13-L17) – essentially the same. Again, as expected, normalizer doesn’t help for adversarial attacks since these attackers *didn’t rely on simple obfuscation* (they had more cunning tricks)[[127]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L183-L192). v3 alone was 50.0%, v1 alone a measly 20.4%[[128]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L14-L19). So v1 is even more hopeless here (because a knowledgeable adversary will avoid every keyword v1 looks for) – it caught only 20 of 98 attempts[[129]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L15-L19). v3 and v1+v3 caught ~49-54, so about half. The normalizer had zero effect (53.1 vs 54.1 is within one sample difference)[[125]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L13-L21). They conclude **“Normalizer doesn’t help against adversarial attacks”** – those attacks aren’t using homoglyphs or zero-width chars, they’re using logic and semantic tricks[[127]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L183-L192).
* **Detector evasion attacks:** Only **15%** detected by the ensemble[[130]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L60-L64). v1 caught 0% of these (by design, since these attacks avoid v1’s triggers)[[130]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L60-L64). v3 caught 15%. So 85% of inputs explicitly written to avoid known signatures & rules went through. This is expected – it shows if an attacker knows your signature list, they can *trivially rephrase to defeat v1*. The small 15% caught likely had to be detected by some anomaly that v3 noticed. They label this gap as **critical (85% evasion rate)**[[131]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L60-L68) and note “detectors are vulnerable to targeted evasion”[[131]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L60-L68)[[132]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L66-L67). This aligns with what any security expert would suspect; it’s good they tested it and quantified it.
* **Normalizer evasion attacks:** These were *completely thwarted* – **100% detected by v3/ensemble**[[133]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L68-L76). In fact, v3 caught all 19 of these, v1 caught 68%, but since v3 got 100%, the combo is 100%. This is a **strength**: apparently, attempts to confuse the normalizer *failed*, and once normalized, the attacks had telltale signs that v3 or v1 caught[[133]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L68-L76). For example, an attacker might try a weird homoglyph the normalizer *doesn’t* map – but maybe v3’s Unicode anomaly feature catches it anyway. Or they insert zero-width characters in a novel way – the normalizer likely still strips them. This result shows the normalizer’s design is quite robust to even creative obfuscation. The authors call out *“Normalizer evasion is well-handled (0% evasion rate)”*[[133]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L68-L76)[[134]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L72-L74). This backs the claim that the normalizer is effective at its narrow job.
* **Threshold evasion attacks:** Only **45% detected**, meaning 55% sneaked under the threshold[[135]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L76-L84). Since we know in Phase 5 they tuned for 0 FPs, the threshold is likely high – an attacker can try to craft a prompt that only lightly trips detectors. The results: v1 got 0% (because these probably avoided v1 triggers as well), v3 45%, ensemble 45%[[135]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L76-L84). So more than half of these subtle attacks evaded detection. They term this a **gap** – low-confidence malicious inputs can slip by[[136]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L80-L82). It highlights that a static threshold can be a vulnerability: an attacker can intentionally make their prompt borderline (e.g., asking for disallowed content in a very implicit way so the detector isn’t confident).
* **Feature confusion attacks:** Ensemble caught ~58%[[137]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L84-L92), so 42% evaded. These attacks likely looked very strange (to confuse features) yet not malicious enough for rules. Interestingly v1+v3 did a bit better (63.2%)[[137]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L84-L92), possibly because v1 might catch a few that v3 didn’t when weird patterns are present. Normalizer+v1+v3 was 57.9%, slightly less, possibly due to small sample differences[[137]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L84-L92). The key is ~40% evasion. They mention things like entropy/length manipulation can confuse the model’s features[[138]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L226-L234), which is believable. This is a moderate vulnerability – not as catastrophic as detector evasion, but still substantial.
* **Gradient attacks:** ~50% detected[[139]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L92-L100), so half got through. v1 was very low (10%), v3 45%, ensemble 50%[[139]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L92-L100). This indicates that an attacker using a more systematic approach (perhaps optimizing input to minimize the logistic regression output) can evade half the time. They note this means the *decision boundary itself can be exploited*[[140]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L230-L238) – typical of ML systems, adversaries can find inputs near the boundary that slip by. This category overlaps with threshold evasion but using an algorithmic approach rather than intuition.

Overall, **system TPR ~53%** on adaptive attacks versus ~49% on novel attacks (6b) and 87% on known attacks[[141]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L198-L206). The authors point out the trend: as attacks get more sophisticated (known → novel → adversarial), detection falls off, though interestingly adversarial (targeted) attacks were *slightly* easier to catch than completely novel ones[[142]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L202-L210). They speculate that completely novel attacks (6b) might include some that are very hard for detectors (like multi-turn, context) whereas the adversarial set at least had some that normalizer/v3 could nail (the normalizer-evasion category failed entirely, boosting the average)[[142]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L202-L210). This observation is insightful and shows they are comparing phases to draw higher-level conclusions, which is great in a discussion section.

**Conclusions & Vulnerabilities:** The Phase 6c report clearly identifies **critical vulnerabilities**: (1) Detector evasion and (2) Threshold evasion (85% and 55% evasion rates respectively)[[143]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L213-L222). These are basically the semantic gap issues – an attacker who rephrases or hides intent can bypass current detectors. **Moderate vulnerabilities**: (3) Feature confusion (42% evasion) and (4) Gradient attacks (50% evasion)[[144]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L225-L234). And a **strength**: (5) Normalizer evasion is not effective (0% evasion)[[145]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L237-L243). This list is essentially a security audit of the system, and it’s very valuable. They’re essentially doing a mini threat modeling exercise by enumerating what an attacker can do and how successful it would be. No sugarcoating: e.g., “Detector evasion: 85% evasion rate – attackers can craft queries avoiding specific keywords, paraphrasing breaks v1, etc.”[[146]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L215-L224). This level of detail is exactly what a critical reviewer would hope to see, because it shows the authors understand where their system can fail.

**Recommendations for Improvement:** They don’t just list problems; they propose fixes prioritized by impact:

* High priority: **Improve detector robustness** (to paraphrasing) – e.g., expand the keyword list, use semantic similarity matching, paraphrase detection (these could boost detector evasion TPR by 30-40%)[[147]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L248-L256). And **Threshold robustness** – calibrate confidence, ensemble scoring, detect when an attack is low-confidence by design (could gain 20-30% TPR)[[148]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L254-L260). These directly address the critical gaps.
* Medium priority: **Feature robustness** – consider adversarial training or more robust features to reduce feature confusion attacks (maybe +15-25% TPR)[[149]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L264-L272). And **Gradient robustness** – e.g., gradient masking or adding noise to decision boundary (could add 10-20% TPR)[[150]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L270-L274). These are more researchy solutions.

By quantifying expected improvement (though those numbers seem like rough guesses), they convey that the system can be substantially hardened, but it would require significant effort (and possibly ML approaches like semantic embedding models, adversarial training, etc.). This shows foresight and prevents a reviewer from saying “you identified problems but what now?” – they already have an improvement roadmap.

**Limitations:** They list similar ones to 6b: these adversarial attacks were synthetic and limited, not exhaustive, and they did not simulate a *fully adaptive attacker* who might iteratively probe the system or have white-box knowledge[[151]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L278-L285). This is fair – building a truly adaptive adversary is beyond the scope here, but acknowledging it means they’re not claiming victory. They basically frame Phase 6c as a static one-shot adversarial evaluation.

**Overall assessment:** Phase 6c confirms the system’s partial effectiveness and clear weaknesses against a knowledgeable attacker. The authors conclude that while the system is *“moderately robust”* (catching ~53%), it has *“clear vulnerabilities”*[[152]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L20-L25). They still recommend using **Normalizer+v1+v3 for production** (since it was best in this scenario, albeit by a hair) but *“with monitoring & improvements planned”*[[153]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L296-L300)[[154]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L314-L318). They emphasize the **key achievement** of Phase 6c was identifying these vulnerabilities and confirming normalizer’s value and v3’s relative strength over v1[[155]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L289-L297).

From a critical reviewer standpoint, Phase 6c is well-done and necessary. It prevents the paper from making any grandiose “we solved it” claims. In fact, it reinforces that *attack-defense is a moving target* – you improve defenses, attackers respond with new tactics, and so on. The authors treat it as such. They don’t shy away from the fact that an intelligent attacker can get past their system roughly half the time. This lends a lot of credibility: it shows the work is realistic about its contributions. The conclusions and future work stemming from this are appropriately measured.

**Publication readiness note:** By the end of Phase 6, the documentation has a full arc: from strong results on known data to frank discussion of limits on novel/adversarial data. This balance is crucial for a publication. All tables and figures (the breakdown tables for 6b and 6c, etc.) support these conclusions with concrete numbers (e.g., 0% vs 15% vs 45% TPR in various categories)[[104]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L77-L85)[[135]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L76-L84). The writing remains clear, and the inclusion of these last phases ensures the paper will not be seen as naive about security.

## Overall Conclusions and Assessment of Readiness

**Logical Soundness:** The experimental logic flows well through the phases. Each phase addresses a question raised by the previous: Phase 1 shows vulnerability → Phase 2/3 build a defense for known attacks → Phase 4 ensures it’s robust in operation → Phase 5 strengthens it with learning and addresses obfuscation → Phase 6 tests it against new threats and edge cases. This progression means the final conclusions are well-supported. For example, the recommendation of *Normalizer+v3 for production* is not arbitrary – it was borne out by Phase 6a (safety on benign) and Phase 6b/c (not much worse detection than using v1, but far fewer false positives)[[156]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/README_FINAL.md#L143-L152)[[78]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L19-L22). The suggestion to use *Normalizer+v1+v3 in monitoring mode* (for higher recall) is justified by the data (it catches ~5% more novel/adversarial attacks at the cost of much higher FAR)[[157]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/README_FINAL.md#L149-L153)[[78]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L19-L22).

**Credibility of Numbers:** All key performance claims in the report are backed by data and often citations of exact figures: e.g., *“Phase 3 baseline v1+v3: 87.0% TPR, 0.0% FAR”*[[158]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/README_FINAL.md#L56-L64), *“Phase 5 (fusion) achieved 99.0% TPR ±2.2%”*[[51]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L13-L21), *“Phase 6a Normalizer+v3 FAR 0.77% (2/260)”*[[64]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L13-L19), *“Phase 6b TPR 49.2% on novel attacks”*[[92]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L13-L21), *“Phase 6c TPR 53.1% on adversarial”*[[125]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L13-L21). These figures are consistent across the documentation and whenever a claim is made, the data supporting it is either shown in a table or cited. The report often provides confidence intervals and significance tests (e.g., Wilson CI for rates, McNemar tests for differences)[[2]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase1/PHASE1_COMPLETION_SUMMARY.md#L39-L44)[[16]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L124-L132) – that level of statistical reporting is above-and-beyond for many security papers, and it increases trust that the results are not flukes. I cross-checked a few: the Phase 5 improvement of +12% TPR is exactly the difference between 87% and 99% they reported[[53]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L116-L124); the Phase 6a FAR of 0.77% corresponds to 2/260, indeed what they observed[[64]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L13-L19). Everything is internally consistent.

A minor note: the distinction between using v2 or v3 in certain phases was a little confusing (since v3’s definition evolved). However, the final **Performance Summary** table in the README nicely consolidates the configs: it shows v1, v3, v1+v3, Normalizer+v3, etc., and their Phase1 TPR, Phase6a FAR, Phase6b TPR[[22]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/README_FINAL.md#L132-L140). It matches the detailed reports (e.g., v1+v3 Phase1 TPR 87%, Phase6a FAR 23.8%, Phase6b TPR 49.2%[[159]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/README_FINAL.md#L134-L140), all of which were stated in the respective sections). Such a summary table is great for publication. It might require explaining acronyms, but it effectively communicates the trade-offs of each configuration.

**Insights & Limitations:** The authors do an excellent job highlighting **strengths** (e.g., high detection on known attacks, zero FPs, normalizer effectiveness) and **limitations** (poor coverage of truly novel attacks, vulnerability to adaptive tactics). They are careful not to over-generalize their success. For example, after Phase 5 one could think “we have 99% detection!” but they immediately show in Phase 6b that “99% of known attacks” does not equal “99% of *all* attacks” – it drops to ~50%. They explicitly mention that result as a goal not met[[100]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L173-L180). There’s no sweeping under the rug; they call out that multi-turn and context-based attacks are missed and require future research[[119]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L200-L209). They also mention that none of the novel/adversarial tests were run on a live model (so they are testing detection in isolation)[[91]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L223-L229)[[160]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L280-L285). In a paper, one might add a note that end-to-end evaluation (attack success *and* detection) would be useful, but given how they structured it (first prove attacks can succeed in Phase 1, then focus on detection alone), it’s acceptable.

**Publication-Ready Aspects:** The documentation is comprehensive. For writing a final paper or report, all necessary components are present:

* **Motivation:** demonstrated by Phase 1 (the problem exists, high ASR).
* **Solution Description:** detectors (v1/v2/v3), normalizer, and fusion model are described with enough detail that one understands their approach[[161]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L34-L42)[[47]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L55-L63). The architecture diagram text in the README is a helpful summary of how input flows through normalizer to detectors to fusion[[162]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/README_FINAL.md#L89-L97)[[163]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/README_FINAL.md#L98-L106). I might suggest including that diagram or a refined version as a figure, as it concisely shows the system design.
* **Evaluation:** split into multiple phases each addressing different questions. It covers detection accuracy on known attacks (with stats significance), effect of threshold, false positives on benign, generalization to new attacks, and adversarial robustness. This essentially maps to typical subsections: performance on training distribution, robustness to noise/obfuscation, generalization, adversarial evaluation. Each is supported by tables/graphs (though I see mostly tables in the markdown; perhaps some plots were generated like the heatmap or ROC curves, which could be included). For example, the Phase 1 heatmap of ASR by evasion type and model was mentioned[[164]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase1/PHASE1_COMPLETION_SUMMARY.md#L28-L37) – including that figure in the paper could be nice to visualize which attacks LLaMA vs Falcon failed on. The threshold sweep plot (flat F1 vs threshold) is another interesting figure showing a basically constant line[[44]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/PHASE4_COMPLETE_SUMMARY.md#L88-L96) – a reviewer would find that notable.
* **Discussion:** The later phases double as a discussion of strengths/weaknesses and future work. They clearly enumerate vulnerabilities and improvements (Phase 6c’s list)[[165]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L213-L224)[[147]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L248-L256). They also have a *Phase 5 Future Work* document (saw in search results), likely earlier planning, but Phase 6c covers most of it.
* **Conclusion:** The final Phase 6c conclusion and the README\_FINAL together provide a conclusion. They basically conclude that the system is effective for known attacks and significantly mitigates the problem, but fails to catch everything, especially novel and adaptive attacks – hence it’s a step forward, not a complete solution. This nuanced conclusion is appropriate for publication: it claims the contributions (a multi-layer defense that *can* stop the majority of prompt injections and greatly reduce risk) while acknowledging that determined adversaries can still succeed sometimes, calling for ongoing improvements.

**Minor Points for Publication:**  
- Ensure terminology is consistent (the paper should clearly define v1, v2, v3 detectors and stick to one naming scheme – currently “v3” was used for both a stat anomaly detector and later a “semantic” approach; perhaps clarify that the final v3 is effectively the learned logistic model acting as a semantic detector). This is editorial.  
- Possibly add a brief note on performance overhead: They did mention latency (<0.1ms per sample for Normalizer+v3)[[166]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/README_FINAL.md#L19-L23), which is basically negligible, and complexity in terms of lines of code. This is good because a reviewer might ask if this detection slows things down. It clearly doesn’t. - If this were a paper, I’d expect a related work section comparing to other approaches (like OpenAI’s moderation or prior research on prompt injection defenses). Not much of that is in the docs (aside from using OpenAI’s mod as a baseline in Phase 3 which failed). Since this is a review of their experiment, it’s fine, but for publication they’ll want to position it relative to others. They do highlight what differentiates their work (input-side detection vs response-side, threshold invariance, etc.)[[6]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L12-L20)[[43]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/PHASE4_COMPLETE_SUMMARY.md#L42-L47), which is great content for the intro/discussion of a paper.

**Final Verdict:** This work appears **publication-ready** in terms of having all necessary evidence, a clear narrative, and a balanced evaluation. The numbers are credible and not cherry-picked; improvements are always backed by data (e.g., “+12% TPR lift” with fusion[[53]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L116-L124)) and drawbacks are candidly disclosed (e.g., “51% of novel attacks evade”[[120]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L232-L240)). The conclusions drawn – such as recommending the simpler model for deployment, and acknowledging the unsolved challenges – show appropriate confidence. They neither undersell (they rightly point out the system’s high precision and its effectiveness on known attacks) nor oversell (they openly talk about what it doesn’t catch). This measured tone lends the work authority.

All tables, figures, and descriptions seem in order. The results are summarized in text and tables, with consistency between them. For instance, multiple sections cite “87% TPR, 0% FAR” as a baseline (Phase 3 & 4)[[38]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/PHASE4_COMPLETE_SUMMARY.md#L16-L24), “0.77% FAR” as the achieved benign false alarm rate (Phase 6a)[[64]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L13-L19), “49.2% TPR” on novel attacks (Phase 6b)[[92]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L13-L21), etc., and these are all cross-validated in their respective detailed analyses. The writing in the reports is clear and precise, which bodes well for turning it into a cohesive paper. I am confident that, with minor editorial polishing, this work can be published and will provide valuable insights to the community about a pragmatic multi-layer approach to prompt injection security and its limitations.

[[1]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/README_FINAL.md" \l "L39-L47) [[4]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/README_FINAL.md#L8-L11) [[22]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/README_FINAL.md#L132-L140) [[45]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/README_FINAL.md#L92-L101) [[46]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/README_FINAL.md#L109-L118) [[81]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/README_FINAL.md#L141-L149) [[156]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/README_FINAL.md#L143-L152) [[157]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/README_FINAL.md#L149-L153) [[158]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/README_FINAL.md#L56-L64) [[159]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/README_FINAL.md#L134-L140) [[162]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/README_FINAL.md#L89-L97) [[163]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/README_FINAL.md#L98-L106) [[166]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/README_FINAL.md#L19-L23) README\_FINAL.md

<https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/README_FINAL.md>

[[2]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase1/PHASE1_COMPLETION_SUMMARY.md#L39-L44) [[3]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase1/PHASE1_COMPLETION_SUMMARY.md#L40-L44) [[5]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase1/PHASE1_COMPLETION_SUMMARY.md#L134-L142) [[58]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase1/PHASE1_COMPLETION_SUMMARY.md#L42-L44) [[164]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase1/PHASE1_COMPLETION_SUMMARY.md#L28-L37) PHASE1\_COMPLETION\_SUMMARY.md

<https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase1/PHASE1_COMPLETION_SUMMARY.md>

[[6]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L12-L20) [[7]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L16-L19) [[8]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L35-L44) [[9]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L46-L54) [[10]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L52-L60) [[11]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L62-L70) [[12]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L65-L73) [[13]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L64-L72) [[14]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L83-L91) [[15]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L95-L103) [[16]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L124-L132) [[17]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L85-L91) [[18]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L86-L89) [[19]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L114-L122) [[20]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L109-L117) [[21]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L134-L143) [[161]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md#L34-L42) PHASE2\_INPUT\_DETECTION\_SUMMARY.md

<https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE2_INPUT_DETECTION_SUMMARY.md>

[[23]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L12-L20) [[24]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L16-L24) [[25]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L20-L22) [[26]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L34-L40) [[27]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L16-L20) [[28]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L1-L8) [[29]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L14-L22) [[30]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L24-L32) [[31]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L50-L59) [[32]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L61-L69) [[33]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L138-L141) [[34]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L126-L134) [[35]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md#L50-L58) PHASE3\_CORRECTED\_SUMMARY.md

<https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/PHASE3_CORRECTED_SUMMARY.md>

[[36]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/PHASE4_COMPLETE_SUMMARY.md#L4-L12) [[37]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/PHASE4_COMPLETE_SUMMARY.md#L14-L22) [[38]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/PHASE4_COMPLETE_SUMMARY.md#L16-L24) [[39]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/PHASE4_COMPLETE_SUMMARY.md#L28-L36) [[40]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/PHASE4_COMPLETE_SUMMARY.md#L30-L38) [[41]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/PHASE4_COMPLETE_SUMMARY.md#L12-L19) [[42]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/PHASE4_COMPLETE_SUMMARY.md#L26-L34) [[43]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/PHASE4_COMPLETE_SUMMARY.md#L42-L47) [[44]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/PHASE4_COMPLETE_SUMMARY.md#L88-L96) PHASE4\_COMPLETE\_SUMMARY.md

<https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/PHASE4_COMPLETE_SUMMARY.md>

[[47]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L55-L63) [[48]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L30-L39) [[49]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L36-L44) [[50]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L146-L152) [[51]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L13-L21) [[52]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L77-L85) [[53]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L116-L124) [[54]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L118-L124) [[55]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L19-L22) [[56]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L98-L106) [[57]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L138-L143) [[59]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L76-L84) [[60]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L11-L19) [[61]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md#L19-L25) PHASE5\_ZERO\_FPR\_REPORT.md

<https://github.com/carlosdenner-videns/prompt-injection-security/blob/410097210ed4482a4558cef0887ec054357adea7/phase5/PHASE5_ZERO_FPR_REPORT.md>

[[62]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L28-L37) [[63]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L30-L39) [[64]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L13-L19) [[65]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L200-L208) [[66]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L80-L88) [[67]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L15-L18) [[68]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L57-L65) [[69]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L68-L76) [[70]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L90-L98) [[71]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L61-L64) [[72]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L120-L128) [[73]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L144-L153) [[74]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L59-L64) [[75]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L84-L92) [[76]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L156-L165) [[77]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L13-L21) [[78]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L19-L22) [[79]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L220-L228) [[80]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L180-L188) [[82]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L40-L48) [[83]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L210-L218) [[84]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L104-L113) [[85]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L124-L128) [[86]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md#L186-L194) PHASE6A\_OBFUSCATION\_BENIGN\_REPORT.md

<https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6a/PHASE6A_OBFUSCATION_BENIGN_REPORT.md>

[[87]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L33-L41) [[88]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L33-L40) [[89]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L35-L40) [[90]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L39-L41) [[91]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L223-L229) [[92]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L13-L21) [[93]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L22-L25) [[94]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L113-L121) [[95]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L14-L18) [[96]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L14-L19) [[97]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L50-L58) [[98]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L159-L167) [[99]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L170-L178) [[100]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L173-L180) [[101]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L61-L69) [[102]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L69-L77) [[103]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L75-L76) [[104]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L77-L85) [[105]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L83-L91) [[106]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L85-L93) [[107]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L89-L91) [[108]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L93-L101) [[109]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L97-L100) [[110]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L101-L107) [[111]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L142-L151) [[112]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L154-L162) [[113]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L132-L140) [[114]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L134-L140) [[115]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L185-L193) [[116]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L192-L200) [[117]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L199-L207) [[118]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L210-L218) [[119]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L200-L209) [[120]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L232-L240) [[121]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L234-L242) [[122]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md#L236-L244) PHASE6B\_ADAPTIVE\_ATTACK\_REPORT.md

<https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6b/PHASE6B_ADAPTIVE_ATTACK_REPORT.md>

[[123]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L31-L40) [[124]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L39-L41) [[125]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L13-L21) [[126]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L13-L17) [[127]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L183-L192) [[128]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L14-L19) [[129]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L15-L19) [[130]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L60-L64) [[131]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L60-L68) [[132]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L66-L67) [[133]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L68-L76) [[134]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L72-L74) [[135]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L76-L84) [[136]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L80-L82) [[137]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L84-L92) [[138]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L226-L234) [[139]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L92-L100) [[140]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L230-L238) [[141]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L198-L206) [[142]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L202-L210) [[143]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L213-L222) [[144]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L225-L234) [[145]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L237-L243) [[146]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L215-L224) [[147]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L248-L256) [[148]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L254-L260) [[149]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L264-L272) [[150]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L270-L274) [[151]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L278-L285) [[152]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L20-L25) [[153]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L296-L300) [[154]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L314-L318) [[155]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L289-L297) [[160]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L280-L285) [[165]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md#L213-L224) PHASE6C\_ADVERSARIAL\_ROBUSTNESS\_REPORT.md

<https://github.com/carlosdenner-videns/prompt-injection-security/blob/0a9669c277a41f5458428819aed7c8c8b1efd326/phase6c/PHASE6C_ADVERSARIAL_ROBUSTNESS_REPORT.md>