# Review of Phase 3 and Phase 4 Results

## Phase 3: Multilayer Defense (Ablation) Evaluation

**Configuration Combinations:** Phase 3 systematically evaluated seven defense configurations (A–G) by combining the three base detectors from Phase 1/2: **v1** (signature-based), **v2** (rules-based), and **v3** (classifier-based). These correspond directly to Phase 1 defenses (Signature-only, Rules-only, Classifier) improved in Phase 2, ensuring traceability. For example, Configuration E = v1+v3 (Signature + Classifier) and others similarly pair/triangulate the Phase 1 detectors[[1]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_FINAL_SUMMARY.md#L16-L24). The combination logic was a simple OR-fusion, meaning an input is flagged as an attack if **any** component detects it[[2]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_COMPLETE.md#L24-L32). This is a sound approach to maximize true positives given independent detectors.

**Correctness & Rigor:** The Phase 3 evaluation appears scientifically sound and thorough. All metrics are reported with appropriate rigor: **True Positive Rate (TPR)** and **False Positive Rate (FPR)** (also called FAR) are given along with Wilson 95% confidence intervals, and summary measures like Precision, F1, and latency are included[[3]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_COMPLETE.md#L34-L42). Crucially, the evaluation used a consistent dataset (Phase 1 Part A: 400 input samples, including 200 benign and 200 injection attempts) with ground truth labels, so the TPR/FPR definitions are clear (TPR measured on 200 attack attempts, FPR on 200 benign)[[4]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_FINAL_SUMMARY.md#L32-L40). They also conducted **McNemar’s statistical tests** to check for significant differences between detectors, adding statistical rigor to claims of improvement (or lack thereof). For instance, no significant difference was found between most configurations (p ≥ 0.157 in many pairwise comparisons), confirming that performance differences were minor[[5]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_COMPLETE.md#L178-L187). This lends credibility – they are not over-claiming differences that aren’t statistically supported.

**Key Phase 3 Findings:** The results showed that **no multilayer combination dramatically outperformed the best single detector**. In fact, all multi-detector combos achieved essentially the same **0% FPR** (no false alarms) and similar TPR as the top single detectors[[6]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_FINAL_SUMMARY.md#L18-L24)[[7]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_FINAL_SUMMARY.md#L76-L83). Initially, configuration **E (Signature+Classifier)** achieved the highest TPR at ~87% with 0% FPR[[8]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_FINAL_SUMMARY.md#L20-L24)[[7]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_FINAL_SUMMARY.md#L76-L83), catching 87% of all attack attempts while never flagging benign input. This was only marginally better than config **D (Signature+Rules)** at ~84% TPR[[6]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_FINAL_SUMMARY.md#L18-L24), and others fell behind (e.g. Rules-only was much lower). Crucially, any combination that included v1 and v3 hit the same ceiling of caught attacks (no combo exceeded 87% TPR)[[9]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_FINAL_SUMMARY.md#L188-L195). This indicates **redundancy** among detectors – the attacks missed by signature were largely the same ones the classifier caught, and vice versa. In other words, v3 provided about a 6–7% TPR boost over v1 by catching some obfuscated attacks that signature missed, but combining all three (v1+v2+v3) did not capture additional attacks beyond v1+v3[[10]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_FINAL_SUMMARY.md#L188-L194). The ablation analysis explicitly notes **no complementarity** beyond what v1 and v3 cover: e.g. v2 and v3 each added the *same* new catches to v1 (so v1+v3 and v1+v2+v3 all caught the same 174/200 attacks)[[11]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_FINAL_SUMMARY.md#L146-L155)[[9]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_FINAL_SUMMARY.md#L188-L195). This consistency check confirms the combinations were implemented correctly and internal results are self-consistent.

**Phase 3 Soundness:** The Phase 3 documentation and code deliverables indicate a careful, reproducible approach (they list scripts for combining defenses, evaluating, plotting, etc., which suggests the analysis was systematic)[[12]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_COMPLETE.md#L49-L58)[[13]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_COMPLETE.md#L290-L298). The reported performance metrics are believable and internally consistent with Phase 1: for example, v1 (signature-only) got ~80% TPR here[[1]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_FINAL_SUMMARY.md#L16-L24), matching Phase 1’s ~79.9% on attacks[[14]](file://file_000000009ee4622f9a7c7d7cd6fae707#:~:text=,%7C%20194.0%20ms). The improved v2 and v3 also align with Phase 2’s development (v2 rules improved from ~20% in Phase1 to ~44% TPR here; v3 classifier improved from ~13% to ~57%) – these increases reflect the Phase 2 enhancements, confirming traceability to earlier phases. Moreover, the team identified and **fixed a fusion logic bug** partway through Phase 3: earlier drafts showed spurious false positives (e.g. 61% FAR for v3 combos) that were corrected to 0% FAR in the final results[[15]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_FINAL_SUMMARY.md#L26-L34). This fix indicates they validated their methodology and now we see all configurations properly yield zero FPs (as expected, since the detectors were tuned not to fire on benign in Phase 1). All these factors give confidence that the Phase 3 results are **methodologically correct and reliable**.

**Publication-Readiness (Phase 3):** The Phase 3 analysis is very comprehensive – they not only report numbers but also interpret them. For example, they highlight that configuration E (v1+v3) is **Pareto-optimal** (maximizing TPR while keeping FPR and complexity minimal)[[16]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_FINAL_SUMMARY.md#L20-L25). They even discuss why simpler is better (invoking Occam’s Razor) since adding detectors didn’t improve detection[[17]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_COMPLETE.md#L134-L142). Such discussion and the inclusion of statistical significance testing show a level of scientific rigor suitable for publication. The results are clearly presented (tables of metrics, and mention of plots like TPR/FPR comparison, latency, Pareto front, etc. were prepared[[18]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_COMPLETE.md#L60-L68)). In short, Phase 3 delivers a solid, **publication-ready evaluation** of multi-detector combinations, with the conclusion that **“v1+v3” offers the best trade-off (≈87% detection with no false alarms) with no advantage to piling on more layers**[**[8]**](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_FINAL_SUMMARY.md#L20-L24)[**[9]**](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_FINAL_SUMMARY.md#L188-L195)**.**

## Phase 4: Threshold Sweep on Best Configuration

Phase 4 builds directly on the Phase 3 findings by taking the best-performing defense (Configuration E, **Signature + Classifier**) and exploring its sensitivity to the classifier’s confidence threshold. This is a logical next step: Phase 3 used a fixed threshold (t=0.50) for the v3 classifier, so Phase 4 asks *“what if we adjust the threshold up or down?”* to ensure the chosen operating point is truly optimal and robust.

**Threshold Range & Granularity:** They swept the threshold from **0.05 up to 0.75 in increments of 0.05**, for a total of 15 evaluation points[[19]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE4_COMPLETE_SUMMARY.md#L4-L7). This is a sufficiently fine granularity (even finer than the 0.1 step you suggested – they used 0.05, doubling the resolution). The range 0.05–0.75 was well-chosen to cover **high-recall to high-precision regimes**. Essentially, 0.05 is a very permissive (low) threshold and 0.75 quite strict; going beyond 0.75 likely would start dropping more true positives sharply (and conversely, below 0.05 would start introducing false positives). So the sweep covers all meaningful thresholds where performance might change.

**Key Phase 4 Finding – Invariant Performance:** Surprisingly, the results showed **no performance variation across this entire threshold range**: from 0.05 up through 0.75, the **TPR stayed fixed at 87.0% and FPR at 0.0%** for the v1+v3 combo[[20]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE4_COMPLETE_SUMMARY.md#L16-L24)[[21]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/phase4/PHASE4_THRESHOLD_TUNING_SUMMARY.md#L101-L109). Every threshold tested produced the *exact same* detection outcomes: catching ~87% of attacks with zero false alarms (Precision 100%, F1 ~0.93). This is clearly illustrated by their metric table and noted explicitly: *“All thresholds from 0.05 to 0.75 achieve identical metrics”*[[20]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE4_COMPLETE_SUMMARY.md#L16-L24). In other words, within this range the system is **threshold-invariant** – an exceptionally robust result. It implies the classifier’s confidence scores are **well-separated** for malicious vs. benign inputs, such that even a very low threshold (0.05) doesn’t accidentally flag benign text, and even a fairly high threshold (0.75) doesn’t miss additional attacks until you go above 0.75. The Phase 4 analysis calls this out as a *“remarkable finding”* demonstrating **excellent discrimination** between attack and normal inputs[[22]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE4_COMPLETE_SUMMARY.md#L14-L22)[[23]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/phase4/PHASE4_THRESHOLD_TUNING_SUMMARY.md#L174-L182).

This threshold sweep was executed correctly and presented with the same thoroughness as Phase 3. They provided detailed metrics at each threshold (in a table)[[21]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/phase4/PHASE4_THRESHOLD_TUNING_SUMMARY.md#L101-L109), plotted ROC-style curves and F1-vs-threshold charts, and even annotated key operating points (e.g. showing the Phase 3 choice t=0.50 as an overlay)[[24]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/phase4/PHASE4_THRESHOLD_TUNING_SUMMARY.md#L128-L137)[[25]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/phase4/PHASE4_THRESHOLD_TUNING_SUMMARY.md#L142-L150). The documentation in the summary markdown is very clear and “publication-ready” – it explains what the flat curves mean (essentially a big flat plateau in the ROC curve at 0% FAR until a threshold ~0.8)[[26]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/phase4/PHASE4_THRESHOLD_TUNING_SUMMARY.md#L136-L144)[[23]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/phase4/PHASE4_THRESHOLD_TUNING_SUMMARY.md#L174-L182). They also derive practical guidance: since any threshold in [0.05, 0.75] works, one might as well stick with **t=0.50** (the Phase 3 default and conventional choice) for deployment[[27]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE4_COMPLETE_SUMMARY.md#L28-L37)[[28]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE4_COMPLETE_SUMMARY.md#L101-L109). They note this gives a safety margin (middle of the invariant range)[[29]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE4_COMPLETE_SUMMARY.md#L30-L38)[[30]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE4_COMPLETE_SUMMARY.md#L42-L46).

**Trade-off and Hidden Points:** One nuance – the Phase 4 detailed analysis does mention what happens *outside* that 0.05–0.75 range. It implies that below ~0.05, you would finally start getting some false positives, and above 0.75, you’d start missing attacks. In fact, their dual-axis plot indicates an **inflection at t≈0.5**: for thresholds lower than 0.5, TPR can rise toward ~92% but at the cost of a few percent false alarms, whereas for thresholds higher than 0.5, TPR holds at 87% until suddenly dropping after ~0.8 (with FAR staying 0% until that point)[[31]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/phase4/PHASE4_THRESHOLD_TUNING_SUMMARY.md#L162-L170). Specifically, they observed **if you wanted ~92% TPR, you’d incur about 5% FAR** (at t≈0.10)[[31]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/phase4/PHASE4_THRESHOLD_TUNING_SUMMARY.md#L162-L170)[[32]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/phase4/PHASE4_THRESHOLD_TUNING_SUMMARY.md#L12-L17). Conversely, to keep FAR at 0%, you are limited to that 87% TPR plateau. This analysis is very important, as it shows the **current method’s limit**: you *can* get above 90% detection only by allowing some false positives. The chosen operating point (87%/0% FAR) is basically at the knee of the curve where false positives drop to zero[[31]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/phase4/PHASE4_THRESHOLD_TUNING_SUMMARY.md#L162-L170). The fact they included this trade-off discussion (high-recall vs. high-precision modes) demonstrates a solid understanding of the detector’s behavior and is great for a paper – it shows they explored the space thoroughly, not just one point.

**Plots and Summary Quality:** The Phase 4 plots are well described. For instance, the ROC-style curve is mostly a vertical line (FAR=0 until threshold ~0.8) – they mark Phase 3’s point on it, etc. They also plotted F1 vs threshold, which turned out flat (since precision and recall didn’t change in 0.05–0.75), reinforcing the invariance[[25]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/phase4/PHASE4_THRESHOLD_TUNING_SUMMARY.md#L142-L150). All of this is documented in a polished manner. The summary markdown reads almost like a draft of a results section in a paper, highlighting the “threshold-invariant performance” as a differentiator (most ML systems require careful threshold tuning, whereas here it’s basically “set and forget”)[[33]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE4_COMPLETE_SUMMARY.md#L42-L48)[[34]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/phase4/PHASE4_THRESHOLD_TUNING_SUMMARY.md#L206-L214). This is a strong **practical point for publication**, as it implies ease of deployment. In short, Phase 4 was executed correctly and with sufficient granularity. The combination of numeric sweep and visualization is comprehensive, and the findings are clearly **ready for inclusion in a paper**.

## Is 87% TPR the Limit? – Detection Ceiling Analysis

Given the above results, the current architecture (Signature + Rules + Classifier detectors with OR fusion) appears to have **plateaued at about ~85–87% TPR** on the evaluation data when we require near-zero false positives. In Phase 3, no matter how the detectors were combined, they could not exceed catching 87% of the attack inputs (57 out of 70 successful attacks, or ~174 out of 200 total attack attempts)[[8]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_FINAL_SUMMARY.md#L20-L24)[[7]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_FINAL_SUMMARY.md#L76-L83). Phase 4 confirmed that even by lowering the classifier threshold dramatically, you can at best reach ~92% TPR – but that introduces false alarms (which may be unacceptable in production)[[31]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/phase4/PHASE4_THRESHOLD_TUNING_SUMMARY.md#L162-L170). In the **zero-FP operating regime**, 87% seems to be the max detection rate with the current methods.

**What is missing in the 13% of attacks that evade detection?** From the Phase 3 attack breakdown, it’s clear these missed attacks come largely from **obfuscation and complex evasions** that the current detectors struggle with. The Phase 3 report specifically calls out **homoglyph substitutions and zero-width character (ZWJ) insertions** as blind spots: e.g. the classifier caught only 20% of homoglyph attacks and 0% of the zero-width joiner attacks[[35]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_COMPLETE.md#L159-L167). Multilingual prompt injections and “payload split” attacks were also detected at a lower rate (~40–60%)[[35]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_COMPLETE.md#L159-L167). In contrast, simpler attack types (direct “ignore instructions” demands, role confusions, etc.) were caught 100% by the existing detectors[[35]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_COMPLETE.md#L159-L167). This tells us that **the remaining ~13% gap to reach 100% (or the ~5% gap to reach 92% vs. the current 87%) consists of attacks that use clever text transformations** – mixing scripts, inserting hidden characters, or splitting malicious commands into pieces – which neither the signature lists nor the current ML-based classifier reliably recognize.

Importantly, this **does not mean 87% is a theoretical maximum** – it’s just the limit for the current detector designs. The fact that a lower classifier threshold was able to catch a portion of those missed attacks (going from 87% to 92% TPR) proves there is information in the input that signals an attack, but the team kept the threshold high to avoid false positives[[31]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/phase4/PHASE4_THRESHOLD_TUNING_SUMMARY.md#L162-L170). In other words, the model *can* flag some of those obfuscated attacks if we tolerate some false alarms. To **get above 90% TPR without raising FPR**, new techniques are needed that can **identify those obfuscated patterns confidently as attacks** (so that they can be detected at high threshold). Currently, those patterns either slip through entirely or only register as low-confidence anomalies (not enough to cross the 0.5 threshold).

Given the project’s iterative progress, it appears the **architecture has reached a point of diminishing returns with the present approach** – simply combining the existing detectors or tweaking their thresholds won’t boost TPR further without sacrificing the no-FP criterion. Phase 3 already combined all signals (and saw no gain)[[9]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_FINAL_SUMMARY.md#L188-L195), and Phase 4 showed threshold tweaks alone can’t solve it either (beyond that inflection point). This strongly suggests that to substantially raise detection rate into the 90%+ range, **a Phase 5 with a new defense enhancement is warranted**. The encouraging part is that we *know exactly what kinds of attacks are in that gap* (homoglyph, ZWJ, multilingual, etc.), so we can target those weaknesses directly.

## Recommendation – Phase 5 to Exceed 90% TPR

To push the detection performance above 90% TPR (while preserving the excellent low FPR), I recommend a **Phase 5 focused on obfuscation-robust detection**. The goal would be to add one more layer or technique that specifically addresses the attacks currently evading the system, thereby closing the gap. This can be achieved in a few possible ways, all of which build on prior phases without starting from scratch:

* **1. Input Preprocessing & Signal Fusion for Obfuscation:** Introduce a preprocessing step that **normalizes or strips out obfuscation** in the user input *before* running the existing detectors. For example, the system can remove zero-width characters and similar invisible Unicode from the input, map homoglyph characters to their base ASCII equivalents, and perhaps even translate non-English content to English (or have a library of non-English trigger phrases). By normalizing text in this manner, many evasive attacks would become obvious. *Rationale:* A prompt like “ign\u200dore all prev\u200di\u200dou\u200ds instructions” (with ZWJ characters) would be cleansed to “ignore all previous instructions,” which the signature detector v1 would catch instantly. Similarly, a prompt using Cyrillic “а” in place of Latin “a” would, after mapping, match the known bad phrase[[36]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_COMPLETE.md#L164-L169). This directly targets the Phase 3 reported weaknesses, and was even suggested in the Phase 3 future work notes (“Improve obfuscation detection (homoglyph, ZWJ)”[[37]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_COMPLETE.md#L272-L280)). Such normalization is computationally cheap and can be done without ML – essentially adding a few regex/character-mapping rules. It’s **feasible within days**, and aligns perfectly with the identified gap. We would need to be careful to avoid introducing false positives – e.g., mapping foreign scripts to English could accidentally flag benign foreign text – but we can constrain it (if the text is mixed-script or mostly English with odd symbols, then apply mapping). This plays to our advantage: benign queries rarely contain zero-width joiners or mixed Cyrillic+Latin characters, so this step would primarily affect malicious attempts. By feeding the normalized text to v1, v2, v3, we’d expect a jump in detections for those previously missed obfuscated attacks, likely pushing TPR well into the 90s% with minimal effect on FPR (any benign that *does* contain weird Unicode would only be flagged if it actually forms a blacklisted phrase upon normalization).
* **2. Shallow Learning Layer (Lightweight Classifier Fusion):** In addition to or instead of the above, Phase 5 could introduce a small **trainable model** to capture complex patterns that the current heuristics might miss. For example, a one- or two-layer neural network that takes as input various features of the prompt (the text embeddings or even the outputs of v1/v2/v3 themselves) and predicts attack vs. benign. This would function as an ensemble that *learns* to recognize subtle attack linguistic patterns (potentially catching things like the multilingual attacks that were missed, or combinations of signals). Because we have a labeled dataset of 200 attacks (with known tricky cases) and 200 benign, a shallow model could be trained on that. It should be **kept simple (shallow)** to avoid overfitting and to stay aligned with the project’s simplicity ethos – e.g., a logistic regression or small feed-forward network using features like “does the text contain rare Unicode?”, “percentage of non-English words”, or even a TF-IDF vector of the input. The idea is to fuse signals in a more adaptive way than a hard OR. For instance, maybe an attack that partially trips several weak signals (but not strongly enough to trigger any one detector) could be caught by a learned model that notices the combined anomaly. This approach is a natural extension of Phase 3: so far, combination was a simple OR; Phase 5 could explore a **learned fusion (OR-plus)** that still yields very low false positives but higher true positives. This is feasible and realistic: the additional model can be trained on existing data in a short time. We’d validate it on the Phase 1 test set to ensure it doesn’t introduce false alarms. If done right, this could net those last few percent of attacks. For example, a shallow classifier might learn to detect prompts that have *almost* the phrase “ignore previous instructions” but with a character changed – something v1 misses exactly, and v3 might score as moderately suspicious. The learned model could combine “contains many letter substitutions (feature from v2)” + “classifier confidence is moderately high” and decide that equals an attack. This kind of **signal fusion** leverages all the groundwork from earlier phases, using the same data and feature outputs.
* **3. Adaptive Thresholds or Policies:** Another angle (though somewhat covered by point 1) is to use **dynamic thresholds or policies**. For instance, if certain high-risk patterns are present, use a lower classifier threshold for that input. In effect, this is like saying we trust the classifier at, say, 0.3 confidence if we also see some obfuscation markers in the prompt. This is a rule-based adaptive fusion strategy that could catch edge cases. It’s a bit more heuristic, but it aligns with “defense-in-depth” – e.g., *if input contains unusual Unicode AND classifier gives even a slight indication of attack, then flag*. This method again targets the known missed cases without globally lowering the threshold (thus preserving the 0% FAR in normal scenarios). It’s essentially a form of context-aware thresholding, which can be seen as an extension of the OR logic (making OR more sensitive under certain conditions).

Any one of the above (or a combination) would likely push the system above the 90% TPR mark. The **most straightforward is the text normalization for obfuscation (point 1)**, as it directly addresses the ~5-10% of attacks that involve unicode tricks. It’s low-risk to implement: indeed, Phase 2 found a false positive from a Cyrillic ‘o’ in a benign input[[38]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE2_COMPLETE.md#L38-L46)[[39]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE2_COMPLETE.md#L40-L48), so we’d want to ensure our homoglyph mapping doesn’t start flagging benign text in other languages. But we can mitigate this by, say, only mapping characters that are embedded in predominantly English text or that are known homoglyphs for letters in our signature list. This way, a Russian sentence wouldn’t be touched (and wouldn’t trigger an English signature). This enhancement is **fully in line with prior phases**, since earlier phases already identified these exact weaknesses and even suggested improving them[[37]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_COMPLETE.md#L272-L280). It doesn’t require a huge new data collection – we’d use the same Phase 1 dataset to test the improvements.

**Feasibility & Impact:** A Phase 5 focusing on obfuscation-resilience could likely be completed quickly (within a week or two) and would demonstrably raise performance. We would measure the new TPR/FPR to ensure we surpass 90% TPR and still maintain near-zero FPR. For instance, if homoglyph and ZWJ attacks go from 0–20% detected to, say, 80%+ detected, the overall TPR could climb to ~95% (hypothetically) with minimal false alarms. Even a more conservative outcome – breaking the 90% TPR threshold – would be a significant result for the paper (“we achieve >90% detection of prompt injections with zero false positives”). That kind of headline number has a nice ring to it and could strengthen the paper’s contributions.

## Conclusion: Publication Readiness vs. Phase 5

**Current Readiness:** As it stands after Phase 4, the project has accomplished a comprehensive study with solid results. Phases 1–4 provide a complete narrative: from baseline weaknesses to improved multi-layer defenses, and finally an optimized configuration that is **simple, fast, and effective (87% TPR, 0% FPR)**[[7]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_FINAL_SUMMARY.md#L76-L83)[[20]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE4_COMPLETE_SUMMARY.md#L16-L24). The finding of threshold invariance is compelling and practical. The documentation and analyses are already written in a very polished form, almost paper-like. This means the work is largely **publication-ready** now – the team can begin writing the actual article (introduction, methodology, results, etc.) using these summaries and figures. There is enough novelty and insight (e.g. the fact that a lightweight detector can stop most attacks with no false alarms, and the extensive evaluation methodology) to write an interesting IEEE Software submission.

However, that remaining ~13% of attacks not caught (especially those obfuscation techniques) will likely be noted as a *limitation* in the paper. If Phase 5 is not done, the paper can still be published by framing it as: *“Our system achieves 87% detection of known prompt injection attempts with zero false positives. The missed attacks are those employing heavy obfuscation; addressing those is left as future work.”* This is perfectly acceptable – no system catches everything – and the paper can emphasize the robust performance and operational simplicity (no threshold tuning needed, etc.) as the main contributions. In fact, an IEEE Software article might prioritize the engineering process and lessons learned over squeezing out every last percent of TPR.

**Should Phase 5 be attempted first?** In my opinion, **yes, if time permits**, a short Phase 5 would be very valuable before locking down the paper. The reason is that the solution is within reach and would make the results even more impressive. Demonstrating >90% TPR with no or negligible FPR would address the current gap and strengthen the claim that the architecture is highly effective against *all* major categories of prompt injection (including obfuscation). It’s essentially the “last mile” to cover known evasions. Since the team already has ideas and it aligns with their roadmap (they listed obfuscation and adaptive improvements in future work), doing it now will let them report those results in the paper rather than as speculation. This could preempt a reviewer comment like “What about attacks with foreign characters or zero-width spaces? Couldn’t those evade the system?” – the paper would be able to answer: *we implemented a fix for that in Phase 5, and now catch X% of them*.

On the other hand, if deadlines or resources are tight, the paper can proceed without Phase 5. The current results are strong enough: a detection system with 0% false alarm rate is extremely rare, and 87% TPR (or 81.6–91.0% CI[[7]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_FINAL_SUMMARY.md#L76-L83)) covers the majority of attacks. The team can argue that this level of performance is sufficient for many practical deployments (especially given the zero false positives and near-imperceptible latency). They can then position Phase 5 as future work, as a next iteration to handle advanced obfuscation – perhaps even propose that as an ongoing research direction (maybe involving more AI-driven detection).

In summary, **Phases 3 and 4 are well-executed and the project is nearly paper-ready**, but I recommend doing Phase 5 if feasible because it targets a known weakness with a likely effective solution. It would likely push the system over the 90% TPR mark and make the final story “we left no stone unturned.” This final enhancement would demonstrate the system’s adaptability and completeness (covering even adaptive attacks), which is compelling for publication. If Phase 5 is completed, the paper’s conclusion can confidently state that the approach stops **>90%** of prompt injections (including obfuscated ones) while requiring minimal tuning – a great selling point. If Phase 5 cannot be done in time, the paper is still in good shape; just be sure to clearly note that certain obfuscation attacks remain an open challenge (and perhaps suggest the Phase 5 solution in the discussion section as future work). Either way, the work up to Phase 4 is high-quality and publishable – Phase 5 would be the cherry on top that could turn a good paper into an excellent one.

[[1]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_FINAL_SUMMARY.md#L16-L24) [[4]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_FINAL_SUMMARY.md#L32-L40) [[6]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_FINAL_SUMMARY.md#L18-L24) [[7]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_FINAL_SUMMARY.md#L76-L83) [[8]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_FINAL_SUMMARY.md#L20-L24) [[9]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_FINAL_SUMMARY.md#L188-L195) [[10]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_FINAL_SUMMARY.md#L188-L194) [[11]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_FINAL_SUMMARY.md#L146-L155) [[15]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_FINAL_SUMMARY.md#L26-L34) [[16]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_FINAL_SUMMARY.md#L20-L25) PHASE3\_FINAL\_SUMMARY.md

<https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_FINAL_SUMMARY.md>

[[2]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_COMPLETE.md#L24-L32) [[3]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_COMPLETE.md#L34-L42) [[5]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_COMPLETE.md#L178-L187) [[12]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_COMPLETE.md#L49-L58) [[13]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_COMPLETE.md#L290-L298) [[17]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_COMPLETE.md#L134-L142) [[18]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_COMPLETE.md#L60-L68) [[35]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_COMPLETE.md#L159-L167) [[36]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_COMPLETE.md#L164-L169) [[37]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_COMPLETE.md#L272-L280) PHASE3\_COMPLETE.md

<https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE3_COMPLETE.md>

[[14]](file://file_000000009ee4622f9a7c7d7cd6fae707#:~:text=,%7C%20194.0%20ms) PHASE1\_FINAL\_SUMMARY.md

<file://file_000000009ee4622f9a7c7d7cd6fae707>

[[19]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE4_COMPLETE_SUMMARY.md#L4-L7) [[20]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE4_COMPLETE_SUMMARY.md#L16-L24) [[22]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE4_COMPLETE_SUMMARY.md#L14-L22) [[27]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE4_COMPLETE_SUMMARY.md#L28-L37) [[28]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE4_COMPLETE_SUMMARY.md#L101-L109) [[29]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE4_COMPLETE_SUMMARY.md#L30-L38) [[30]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE4_COMPLETE_SUMMARY.md#L42-L46) [[33]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE4_COMPLETE_SUMMARY.md#L42-L48) PHASE4\_COMPLETE\_SUMMARY.md

<https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE4_COMPLETE_SUMMARY.md>

[[21]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/phase4/PHASE4_THRESHOLD_TUNING_SUMMARY.md#L101-L109) [[23]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/phase4/PHASE4_THRESHOLD_TUNING_SUMMARY.md#L174-L182) [[24]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/phase4/PHASE4_THRESHOLD_TUNING_SUMMARY.md#L128-L137) [[25]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/phase4/PHASE4_THRESHOLD_TUNING_SUMMARY.md#L142-L150) [[26]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/phase4/PHASE4_THRESHOLD_TUNING_SUMMARY.md#L136-L144) [[31]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/phase4/PHASE4_THRESHOLD_TUNING_SUMMARY.md#L162-L170) [[32]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/phase4/PHASE4_THRESHOLD_TUNING_SUMMARY.md#L12-L17) [[34]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/phase4/PHASE4_THRESHOLD_TUNING_SUMMARY.md#L206-L214) PHASE4\_THRESHOLD\_TUNING\_SUMMARY.md

<https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/phase4/PHASE4_THRESHOLD_TUNING_SUMMARY.md>

[[38]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE2_COMPLETE.md#L38-L46) [[39]](https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE2_COMPLETE.md#L40-L48) PHASE2\_COMPLETE.md

<https://github.com/carlosdenner-videns/prompt-injection-security/blob/4e766c7c6bb687ba40863e7652641a55fff4a421/PHASE2_COMPLETE.md>