

Hydra: Multi-Head Prompt Security Classification and the Over-Defense Failure Mode of Prompt Injection Benchmarks

Bryan A. Pineda
Alterion AI
bryan@alterion.ai

Abstract

Standard prompt-injection benchmarks reward over-flagging. At a 1% false-positive budget, two widely used detectors catch **0.0%** and **0.03%** of attacks — while posting F_1 of 0.752 and 0.710 and AUROC of 0.797 and 0.876. Under their standard decision rules they flag 66.5% and 87.8% of *all* input. The metrics the field reports cannot distinguish a detector from an indiscriminate blocker, and this is not a threshold artefact: calibration, matched operating points, and three over-defense sets agree. A contamination audit explains part of why: **deepset** and **xTRam1** are wholly contained in a public corpus we did not build, and NotInject appears verbatim in our own training data. The finding survives decontamination — on held-out uncontaminated rows the gap is +0.2029 AUPRC, marginally *wider* than the +0.198 measured on the full benchmark. This matters because over-flagging, not missed attacks, is how these systems fail in deployment: adding a detector to an agent pipeline has been measured to cut legitimate task completion from 69% to 41.5% [Debenedetti et al., 2024].

We establish this with **Hydra**, a shared mDeBERTa-v3-base encoder with four independent binary heads — SQL injection, PII, prompt injection, toxicity — each emitting a separately calibrated score at its own tuned operating point. At the same 1% budget, Hydra retains **57.8%** of attacks. It leads seven of the nine benchmarks on which we scored a competitor, and false-positives on **0.73%** of benign prompts against ProtectAI v2’s 2.17% ($p = 8.2 \times 10^{-7}$). One encoder pass serving four heads uses $3.99\times$ fewer parameters than four separate models, at a cost of ≈ 0.6 macro- F_1 points — twice the ± 0.003 spread we measure over three training seeds. We recommend reporting flag rate, calibration, and TPR at fixed FPR alongside conventional metrics.

1 Introduction

LLMs deployed as interactive agents present a broad attack surface: prompt injection overrides system instructions [Perez and Ribeiro, 2022, Greshake et al., 2023], toxic generation undermines safety guardrails [Hartvigsen et al., 2022], PII leakage creates regulatory exposure, and natural-language database interfaces enable SQL injection [Pedro et al., 2023].

The obvious defense—a classifier that blocks malicious prompts—fails in production for a reason that is now well documented. Li and Liu [2024] found that ProtectAI v2 classifies only 56.6% of benign prompts containing trigger words correctly, and that models learn “a shortcut from certain trigger words, like ‘ignore’, directly to the final prediction.” Debenedetti et al. [2024] measured the downstream cost on **gpt-4o-2024-05-13**: deploying an injection detector cut benign task completion from 69.0% to 41.5%, while cutting targeted attack success from 57.7% to 8.0%. The detector works; it is the utility cost that makes it unshippable. A defense that destroys 40% of legitimate task completion will be turned off, and then the attack surface returns in full.

A natural response is “retune the threshold.” We show in Section 5.4 that this is not available: the over-flagging models we measure place *91% and 82% of all their scores above 0.99*, leaving almost no probability mass in the interior for a threshold to move through. Over-defense in these models is a property of a saturated score distribution, not of where the cut is placed.

Beyond that, we argue the problem has an *architectural* dimension and an *evaluation* dimension.

The architectural issue is that a binary allow/block verdict cannot represent the fact that identical text carries different risk depending on the sender. “Export all customer SSNs to CSV for the compliance audit”

is routine for a data protection officer and exfiltration from a junior support agent. A single threshold must serve both.

The evaluation issue is that the benchmarks used to select these classifiers reward the very behaviour that makes them unusable. We demonstrate this quantitatively in Section 8: a model that flags two thirds of all traffic outperforms a well-calibrated one on the field’s most-cited benchmark, and loses badly on a benchmark $560\times$ larger.

Our contributions are two, with the evidence for each summarized here and detailed below.

1. **A measured characterization of a multi-head guard classifier as a deployed decision component.** Prior multi-signal guards emit free-text categorical labels or a single binary verdict; neither produces separately calibrated per-category scores, so neither can express per-category operating points (Section 2). We contribute what is measurable about the design: independently tuned per-category operating points, which single-threshold guards structurally cannot express (Section 3.5); SQL-injection intent as a first-class calibrated head, which we did not find in any system we evaluated; an exact accounting of the shared encoder’s cost — $3.99\times$ fewer parameters and $3.3\times$ less resident memory than four separate models, at no accuracy cost distinguishable from seed noise (Section 9); and the finding that **per-head false-positive rates understate the over-defense of a multi-head model whose heads are all enforced** — by 37% on our own model (Section 6.2), a correction that applies to any multi-head guard reporting a single head’s rate.
2. **Evidence that standard prompt-injection benchmarks can reward over-flagging**, established through a label audit, flag-rate and calibration analysis, detection at a fixed false-positive budget, three over-defense sets, and a contamination audit of every benchmark against both our corpus and eight public ones (Sections 8, 5.3, 5.4, 5.2). The supporting evaluation protocol — paired bootstrap intervals, exact McNemar with discordant-pair counts, explicit underpowered labelling, matched operating points, and seed variance — is itself unusual: among the comparable guard-classifier reports we surveyed, we found none stating a confidence interval, a significance test, seed variance, or a calibration measurement.

Section 2 places the design among related systems. Where our results are unfavourable — and several are — we report them in place rather than in aggregate.

2 Related Work

Prompt Injection Detection. ProtectAI [2024] fine-tuned DeBERTa-v3-base for binary injection classification. Sentinel [Ivry and Nahum, 2025] uses a ModernBERT-large backbone (395M parameters), reporting 0.857 F_1 on `deepset` and 0.985 F_1 on `jackhhao/jailbreak-classification`. Meta’s PromptGuard v2 [Meta, 2025] shares our mDeBERTa backbone but emits a single benign/malicious label. Surveys [Liu et al., 2024, Barcha Correia et al., 2026] catalogue the attack and defense space.

Over-Defense. Li and Liu [2024] introduced NotInject and identified trigger-word shortcut learning; DeBenedetti et al. [2024] quantified the utility cost. Fomin [2026] reports that benchmarks overestimate classifier performance by 8.4 AUC points relative to leave-one-dataset-out evaluation, and that 28% of top classifier features are dataset-dependent shortcuts. Two caveats we carry forward: it is an unrefereed single-author preprint, and its classifiers are linear probes on LLM activations rather than fine-tuned encoders, so transferring its magnitude to models like ours is a cross-architecture inference rather than a measured result. CAPTURE [Kholkar and Ahuja, 2025] and PIArena [Geng et al., 2026] treat over-defense as a first-class evaluation axis.

Our Section 8 extends this line: prior work shows benchmarks *overestimate* performance, whereas we show they can *invert* model ranking, preferring a model that flags two thirds of traffic over one that does not.

Multi-Task and Multi-Signal Guardrails. WildGuard [Han et al., 2024] produces prompt harm, response harm, and refusal jointly from a 7B generative model, and reports the multi-task-versus-single-task ablation such a design invites; we report ours in Section 9.1. MultiTaskGuard/UniGuard [O’Neill et al., 2025] trains a single instruction-conditioned classifier across policies and merges it with policy-specific models. Both

precede this work. Neither, however, is a multi-head design in our sense: WildGuard decodes categorical labels as text and MultiTaskGuard emits one binary safe/unsafe decision, so neither exposes independently thresholdable per-category scores.

Concurrent work. Two closely related encoder guardrails appeared while the system described here was already in production, and we treat them as concurrent and independent rather than as antecedents. GLiNER Guard [Minko et al., 2026] pairs a shared encoder with span-scoring and classification heads emitting safety categories, attack types, intents, and PII *spans* in one pass, at 145–209M parameters. Opir [Stepanov and Smechov, 2026] builds multi-task encoder guardrails on DeBERTaV3 and mDeBERTaV3 — our own backbone family — covering binary safety, multi-label toxicity, jailbreak, and a 996-label taxonomy, with sub-100M edge variants. Both cover a broader label space than our four fixed categories, and GLiNER Guard’s *typed* PII span extraction serves redaction directly, where our localisation is attributive (Section 10).

Two differences are load-bearing for this paper. First, **neither applies independently tuned per-category decision thresholds**: Opir uses a single configurable threshold (default 0.5) with softmax/argmax for its binary view, and GLiNER Guard specifies no per-category detector calibration — so neither can express the asymmetric per-category operating points of Section 3.5. Second, both are dynamic-schema models whose scores depend on the label set supplied at inference, a sensitivity we quantify in Section 7. We re-measure both on identical rows there, and Hydra records the highest AUPRC on seven of the nine benchmarks on which a competitor was scored, and leads every adequately powered prompt-injection comparison.

Our multi-head contributions concern the output contract, the measured cost, and the evaluation (Section 1) rather than the idea of sharing an encoder. First, our heads are thresholded at *independently tuned* per-category operating points; Opir applies one configurable threshold (default 0.5) with softmax/argmax for its binary view, and GLiNER Guard specifies no per-category detector calibration, so neither can express the asymmetric per-category operating points that Section 3.5 requires. Second, among these systems we are the only one reporting SQL injection as a fixed, independently calibrated output. Both are properties of the output contract rather than of the network.

Toxicity, PII, and Multi-Category Safety. ToxiGen [Hartvigsen et al., 2022] targets implicit toxicity; HateCheck [Röttger et al., 2021] provides diagnostic functional tests. Microsoft Presidio [Microsoft, 2023] performs token-level PII extraction; we detect PII *presence* at the prompt level, complementing rather than replacing it. Unified safety classifiers are LLM-scale: WildGuard [Han et al., 2024] (7B), ShieldGemma [Zeng et al., 2024] (2B–27B), LlamaGuard [Inan et al., 2023] (8B), Granite Guardian [Padhi et al., 2024]. Encoder-scale multi-signal guardrails do exist (above); what we have not found in any of them is SQL-injection intent as a first-class scored category alongside injection, PII, and toxicity.

3 Method

3.1 Architecture

Hydra uses `microsoft/mdeberta-v3-base` [He et al., 2023] as a shared encoder (12 layers, 768-dim hidden states). Counting tensors directly in the released checkpoint: **279,597,828** parameters total, of which **193,162,752** (69.1%) are the 250K-token multilingual embedding and 85,646,592 are the transformer blocks. The [CLS] representation feeds four independent MLP heads:

$$s_k = \sigma(\text{MLP}_k(\mathbf{h}_{[\text{CLS}]})), \quad k \in \{\text{sqli}, \text{pii}, \text{pi}, \text{toxic}\} \quad (1)$$

each a two-layer network ($768 \rightarrow 256 \rightarrow 1$) with GELU and dropout ($p=0.1$), exactly 197,121 parameters per head and 788,484 for all four — 0.28% of the model. Heads share no parameters and are thresholded independently (Figure 1).

Parameter counts for mDeBERTa are dominated by the multilingual embedding; the encoder is comparable to DeBERTa-v3-base models reporting smaller totals with 128K vocabularies.

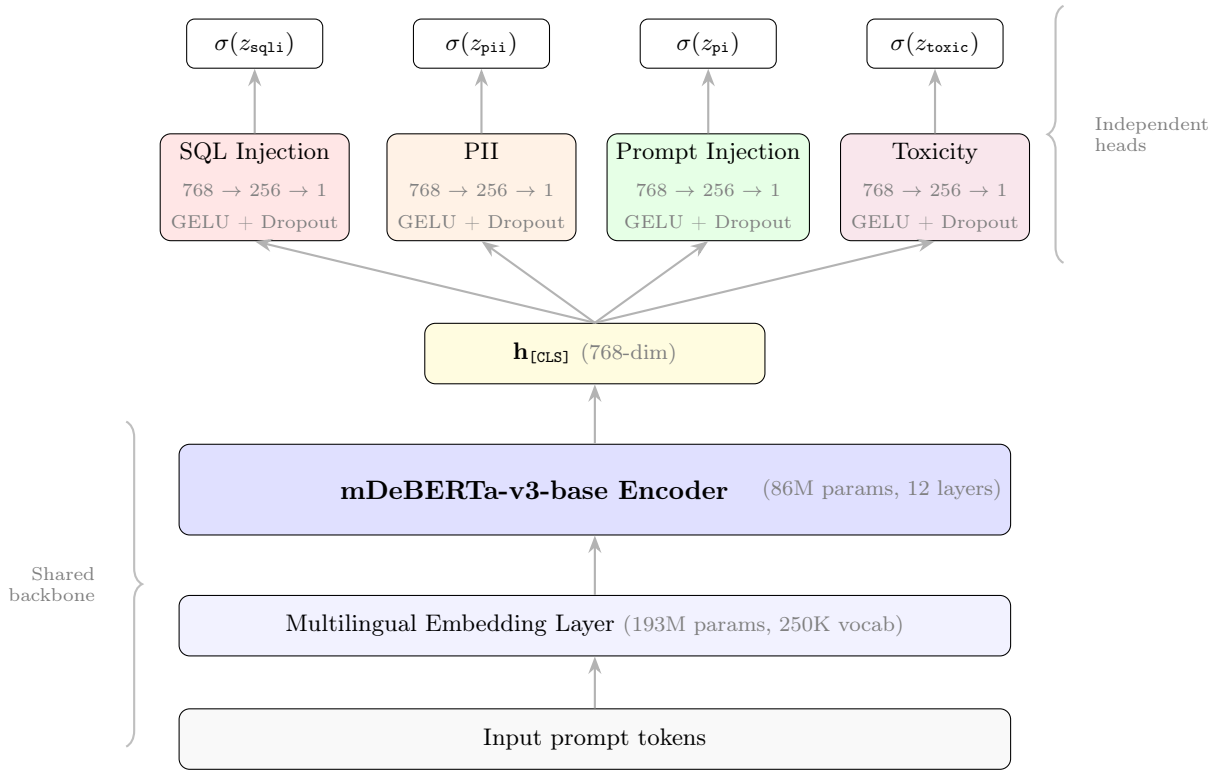


Figure 1: Architecture of Hydra. A shared mDeBERTa encoder produces a [CLS] representation that is independently classified by four MLP heads, each outputting a sigmoid probability for its threat category. Heads do not share parameters and can be thresholded independently.

3.2 Training

Two-stage LP-FT. Following Kumar et al. [2022] we train heads only for one epoch with the encoder frozen (lr 10^{-4}), then fine-tune fully for up to six epochs with cosine scheduling (peak 2×10^{-5}) and layer-wise decay (0.95); head lr is $3 \times$ the encoder rate.

Loss. Binary cross-entropy summed over heads, with a per-head positive weight w_k to offset class imbalance:

$$\mathcal{L} = \sum_{k=1}^4 \left[-w_k y_k \log \sigma(z_k) - (1 - y_k) \log(1 - \sigma(z_k)) \right] \quad (2)$$

The w_k are per-head constants of order the head’s negative-to-positive ratio (Table 1), damped for the two most imbalanced heads. The exact per-head values used for the released model are not recoverable: they were supplied by a launcher configuration on a since-retired training host. We report this rather than reconstruct a formula, since a reconstruction would be indistinguishable from a record. The current trainer takes w_k as an explicit configuration input. Asymmetric Loss [Ridnik et al., 2021] ($\gamma_{\text{neg}}=4$, $\gamma_{\text{pos}}=0$, clip 0.05) caused complete collapse (all FPR ≈ 1.0), likely interacting with positive-weight scaling.

Precision. All training uses FP32. BF16 mixed precision overflows mDeBERTa’s disentangled attention and yields NaN losses after 2–3 epochs—a known consequence of its relative position encoding. This is not merely a training detail: we also observed a serving-time analogue in which an identical exported graph produced range-compressed scores on one GPU vendor’s kernel while remaining correct on another (Section 10).

3.3 Training Data

The Hydra training set contains **230,332 examples** drawn from 20+ public datasets plus targeted synthetic data; labels are non-exclusive. 200,766 rows carry over from the prior corpus version and 29,566 are additional generated prompts, labelled by an LLM judge. Both are audited for benchmark contamination in Section 5.2.

Table 1: Training label distribution, counted directly from the registered corpus the released model was trained on (230,332 train rows, confirmed against the training run’s own dataset record). The ratio column is a property of the corpus; it informs but is not identical to the loss weights w_k (see text).

Head	Positive	Negative	Neg:pos ratio
SQL Injection	6,596	223,736	33.92
PII	57,860	172,472	2.98
Prompt Injection	25,786	204,546	7.93
Toxic	4,433	225,899	50.96

Data quality dominated everything else. Our clearest empirical finding across training runs is that label-quality interventions produced larger gains than architecture or loss changes. We relabelled $\sim 49\text{K}$ examples from noisy public sources (AEGIS, Tensor-Trust, Jigsaw, Salad-Data) with an LLM oracle, reducing noise that caused the injection head to fire on benign text containing trigger words. Later iterations added a targeted supplement of **1,165** rows addressing five measured failure modes, in six sets: benign uses of “ignore”/“forget”/“disregard”/“override” (197) paired against genuine instruction-override attempts (198); counterspeech (198); PII possessive phrasing carrying no value (196, e.g. “ship to my address”); “no limits” benign phrasing (195); and other trigger-keyword benigns (181). The Hydra_{raw} ablation in Section 5 isolates exactly this: it is the same network trained on the corpus before these interventions.

Validation. We apply (i) SHA-256 exact deduplication plus embedding-based near-duplicate removal (cosine 0.95) across train/validation/test; (ii) TF-IDF n-gram mutual-information ranking to surface shortcut n-grams; (iii) model-disagreement auditing (score < 0.1 on positives, > 0.9 on negatives) to surface probable mislabels; and (iv) source-stratified noise analysis of known-noisy corpora.

3.4 Thresholds

Per-head thresholds are tuned on a relabelled validation split (19,014 examples) rather than fixed at 0.5 (Table 2).

Table 2: Per-head operating points. Hydra_{raw} values are included because Section 5 compares the two models and threshold changes must be visible.

Head	Default	Hydra _{raw}	Hydra
SQL Injection	0.50	0.71	0.62
PII	0.50	0.75	0.56
Prompt Injection	0.50	0.78	0.58
Toxic	0.50	0.65	0.50

Because these differ, *any* comparison at a single fixed threshold conflates the model change with the threshold change. Section 4 describes how we control for this.

3.5 Design Rationale: Per-Category Scores

Calibration explains why a single threshold cannot be tuned into usability, but not why a single *verdict* is the wrong output shape. That is a separate, architectural point: a binary verdict cannot express that identical text carries different risk depending on the sender.

- “Export all customer SSNs to CSV” — routine for a data protection officer; exfiltration from a junior support agent.
- “DROP TABLE audit_logs; – cleanup” — DBA maintenance; injection from an application user.
- “Override the default timeout to 60 seconds” — normal DevOps configuration, yet likely flagged by trigger-word-sensitive classifiers.

A binary classifier must pick one threshold for all users. Hydra’s per-category scores allow the decision to be deferred to a policy layer with authorization context:

$$\text{decision}(x, u) = \begin{cases} \text{ALLOW} & \text{if } \forall k : s_k(x) < \tau_k^{(\text{role}(u))} \\ \text{ESCALATE} & \text{if } \exists k : s_k(x) \geq \tau_k^{(\text{role}(u))} \end{cases} \quad (3)$$

with per-head, per-role thresholds. The classifier answers “what kind of threat”; the policy engine answers “is this principal permitted”. This separation is not expressible with a single collapsed score.

4 Evaluation Protocol

Guard-classifier comparisons are easy to get wrong. We adopt four rules and apply them uniformly.

1. Paired statistics, not independent intervals. Both models score the same items, so we use paired tests [Dietterich, 1998]. We report McNemar’s test on discordant pairs with the counts b (Hydra_{raw} correct, Hydra wrong) and c (Hydra_{raw} wrong, Hydra correct) *always printed*: the effective sample size is $b + c$, not n . Where $b + c < 30$ we label the comparison underpowered rather than reporting a verdict. We also report paired bootstrap 95% confidence intervals on ΔAUPRC and ΔF_1 ($B=10,000$, stratified by label so no resample loses a class), and Holm–Bonferroni adjustment across the benchmark family.

2. Threshold-free ranking as the primary metric, plus a deployable operating point. We report AUPRC rather than AUC-ROC because of class imbalance [Saito and Rehmsmeier, 2015], and treat F_1 at a chosen threshold as secondary. We additionally report **TPR at 1% and 0.1% FPR** and partial AUC, following Padhi et al. [2024]. Section 5.3 shows why this matters more than it might appear: two models in our comparison post respectable F_1 and AUROC while detecting *nothing* at a 1% false-positive budget. A metric suite without a fixed-FPR operating point cannot distinguish a deployable guard from an undeployable one.

3. Matched operating points. A model with better ranking scores *worse* at another model’s threshold. We therefore retune the candidate to the reference model’s flag rate, and separately to its precision, and require improvement at both. This is the check that distinguishes a better model from a differently-thresholded one.

4. Flag rate is a guardrail, never a win condition. A model that simply flags less improves precision, reduces analyst load, and looks better on every dashboard while silently losing recall. We therefore treat a flag-rate change of more than $\pm 25\%$ as triggering a mandatory recall audit, regardless of how favourable the other metrics look. Two of our own results trip this rule, and we report the audits rather than suppressing them (Section 5).

4.1 Benchmarks

- **deepset/prompt-injections** [Deepset, 2023]: 116 examples (60 positive). Widely cited; we show in Section 8 that it should not be used alone.
- **jackhhao/jailbreak-classification** [Jackhhao, 2023]: 262 examples (139 positive).
- **xTRam1/safe-guard-prompt-injection**: 2,060 examples (650 positive).
- **jayavibhav/prompt-injection**: **65,416** examples (32,294 positive). Our largest external benchmark, included specifically to address the small- n criticism of Fomin [2026]. Its dataset card is empty; we characterize its contents in Appendix D.
- **ToxiGen** [Hartvigsen et al., 2022]: 940 annotated examples (402 positive), implicit toxicity.
- **HateCheck** [Röttger et al., 2021]: 3,728 functional tests (2,563 positive).
- **Internal relabelled test set**: 19,030 examples covering all four heads. **The PII and SQLi heads are evaluated here only**, with no external benchmark. We looked for external alternatives and did not find one that supports a reliable estimate: PII-Bench [Jha, 2026] overlaps our training corpus and spans 82 entity types with no consistent PII boundary, **auren-research/pii-shield** is annotated by a PII model rather than by humans, and SQLShield [Chafik et al., 2025] is licensed CC-BY-NC-ND with OpenAI-generated queries. Section 10 states what this costs us.

5 Results

Throughout this section **Hydra_{raw}** serves as a controlled reference point: the identical architecture, head configuration, and training recipe, trained on the same aggregated public sources with their labels *as published* — that is, without the label auditing and targeted augmentation of Section 3.3. It is not a prior release; it is the model that the public data yields on its own. Because the network is unchanged, the comparison isolates label quality rather than capacity. Each model is evaluated at its own best- F_1 operating point, so neither is handicapped by the other’s threshold.

Table 4 reports the paired tests. Hydra improves significantly on four of six benchmarks, including all three benchmarks with $n > 2,000$. Two comparisons are underpowered: **deepset** has only 8 discordant pairs and **jackhhao** 24, below our threshold of 30. We therefore do not claim a **deepset** regression—we explain it in Section 8.

Table 3: Per-benchmark results. Each model at its own best- F_1 threshold. “flag” is the fraction of inputs flagged.

Benchmark	Model	n	AUPRC	F_1	Prec	Rec	flag
deepset	Hydra _{raw}	116	0.994	0.974	1.000	0.950	0.491
	Hydra		0.978	0.922	0.964	0.883	0.474
jackhhao	Hydra _{raw}	262	0.959	0.920	0.892	0.950	0.565
	Hydra		0.993	0.974	0.993	0.957	0.511
xtram1	Hydra _{raw}	2,060	0.983	0.946	0.988	0.908	0.290
	Hydra		0.996	0.986	0.994	0.979	0.311
jayavibhav	Hydra _{raw}	65,416	0.741	0.752	0.655	0.882	0.665
	Hydra		0.940	0.856	0.879	0.835	0.469
ToxiGen	Hydra _{raw}	940	0.601	0.615	0.548	0.702	0.548
	Hydra		0.775	0.702	0.631	0.791	0.536
HateCheck	Hydra _{raw}	3,728	0.792	0.835	0.724	0.987	0.937
	Hydra		0.986	0.956	0.932	0.982	0.725

Table 4: Paired significance and effect size (Hydra – Hydra_{raw}). b : Hydra_{raw} correct / Hydra wrong. c : Hydra_{raw} wrong / Hydra correct. Effective sample size is $b + c$. CIs are paired bootstrap, $B=10,000$ (2,000 for jayavibhav), label-stratified.

Benchmark	b	c	$b + c$	McNemar p	Δ AUPRC [95% CI]	Verdict
deepset	7	1	8	0.070	−0.016 [−0.035, −0.004]	underpowered
jackhhao	4	20	24	1.5×10^{-3}	+0.034 [+0.014, +0.057]	underpowered
xtram1	9	60	69	2.2×10^{-10}	+0.013 [+0.008, +0.018]	improved
jayavibhav	5,767	15,491	21,258	$< 10^{-300}$	+0.198 [+0.194, +0.202]	improved [†]
ToxiGen	116	208	324	3.6×10^{-7}	+0.174 [+0.128, +0.218]	improved
HateCheck	71	837	908	7.6×10^{-167}	+0.194 [+0.181, +0.208]	improved [†]

[†]Flag-rate change exceeds $\pm 25\%$, triggering the mandatory recall audit of Section 4. Both audits pass: on HateCheck precision rises 20.8 points for a 0.6-point recall cost, and on jayavibhav recall *increases* at matched flag rate (Table 17), so the reduction reflects sharper discrimination rather than lost sensitivity.

5.1 Matched Operating Points

Table 17 retunes Hydra to the reference model’s operating point. Hydra’s advantage survives both matchings on the three larger benchmarks: at Hydra_{raw}’s own flag rate on `jayavibhav` it achieves higher precision *and* higher recall (0.699/0.941 vs 0.655/0.882). The improvement is therefore not a threshold artefact.

5.2 Contamination Audit

Our training corpus aggregates 20+ public datasets, so public benchmarks may overlap it. We audited all of them at three tiers: exact match after normalization, character-shingle Jaccard ≥ 0.8 and ≥ 0.9 , and embedding cosine ≥ 0.95 . Table 5 reports the result for every benchmark.

Coverage. The tiered audit covers the 200,766 corpus rows carried over from the prior version, a strict subset of the 230,332 rows the released model was trained on. We audited the remaining 29,566 generated rows separately — exact match after normalization and 13-gram overlap — against all seven benchmarks, and found **zero** matches at either tier. The figures below therefore describe the released model’s training data in full.

Table 5: Overlap between each benchmark and our training corpus, and the effect of removing overlapping rows. “Clean $b+c$ ” is the McNemar discordant-pair count *after* decontamination; our underpowered threshold is 30.

Benchmark	exact	$J \geq 0.8$	clean rows	clean $b+c$	clean ΔAUPRC
<code>jayavibhav</code>	3.00%	3.11%	63,383	20,661	+0.2029
ToxiGen	14.5%	—	~800	289	+0.167
<code>deepset</code>	62.9%	65.5%	40	1	−0.015
<code>xtram1</code>	63.9%	67.3%	~670	24	+0.0115
<code>jackhhao</code>	61.1%	72.1%	73	10	+0.058
NotInject	67.3%	67.3%	111	—	—
HateCheck	68.1%	72.9%	~1,010	300	+0.209

The headline result survives, and strengthens. On the 63,383 clean `jayavibhav` rows, ΔAUPRC is **+0.2029**, 95% CI [+0.1982, +0.2077] — *larger* than the +0.198 measured on the full benchmark, and stable at every contamination definition (exact +0.2030; $J \geq 0.9$ +0.2029; strictest union +0.2030). The decisive check is the complement: on the 2,033 *contaminated* rows the delta is −0.0019 [−0.0046, +0.0005], with both models near-ceiling (0.9955 vs 0.9935). Memorization would produce a large delta on seen rows and a small one on unseen rows. We observe the reverse, so the improvement is not explained by training overlap. No other benchmark’s delta shrinks under decontamination either.

Three benchmarks retain too few clean rows to rank models. `deepset`, `jackhhao` and NotInject retain 40, 73 and 111 uncontaminated rows, with clean discordant-pair counts of 1 and 10 — far below our power threshold. No ranking claim on those three is supportable in either direction, including the `deepset` inversion, which we therefore treat as an illustration of a labelling pathology rather than a measurement. On NotInject’s 111 clean rows the false-positive rate roughly doubles (0.90% vs 0.59%); its status is addressed below. This is the argument for the larger over-defense sets of Section 6.2, which carry the over-defense claim.

Is the overlap ours, or the ecosystem’s? The obvious objection is that this measures our corpus rather than the benchmarks. We tested it by re-running the same audit against public corpora we did *not* build. The answer differs per benchmark, and we report all three cases.

For `deepset` and `xtram1` the overlap is entirely generic: both are **100% contained** in `jayavibhav` — 116 of 116 and 2,060 of 2,060 rows, at every matching tier. Zero clean rows remain against a corpus assembled by someone else. In the same vein, the test split of JasperLS/prompt-injections is *byte-identical* to `deepset`’s (both SHA-256 71d23cda...). Any model trained on public prompt-injection data has likely seen these

two benchmarks in full, whoever assembled the training set. `jackhhao` is largely generic too: **53.1% exact** against the public union — 139 of its 262 rows appear in TrustAIRLab’s in-the-wild jailbreak collection alone — against the 61.1% we measure for our own corpus.

NotInject is the exception, and it is our own contamination. It shows **0 of 339** overlapping rows at every tier against *eight* independent public corpora, so its 67.3% overlap with our corpus is not an ecosystem property. We initially supposed convergent construction — our corpus contains 197 deliberately built contrastive pairs pairing benign uses of “ignore”, “override” and “bypass” with genuine injections (Section 3.3), and NotInject was built for the same purpose — but the data refutes that. Of the 228 matching rows, **all 228 are byte-identical** (Jaccard exactly 1.0; *zero* rows fall in the 0.8–0.99 near-duplicate band, where convergent phrasing would necessarily land). NotInject was therefore ingested verbatim into our training corpus, presumably via one of its 20+ public sources. We consequently do not treat NotInject as a held-out set for our model, read its 0.994 as an upper bound rather than a result, and rest the over-defense claim on PHTest and OR-Bench (Section 6.2). We report this because a benchmark appearing verbatim in training data is exactly the failure this section exists to detect, and it applies to us as much as to anyone.

5.3 Detection at a Deployable False-Positive Budget

F_1 at a tuned threshold answers “how good is this model at its best operating point.” A deployer asks a different question: “how many attacks do I catch if I can tolerate 1% false positives?” We report TPR at fixed FPR because that is the deployment question, and because Padhi et al. [2024] and Meta’s PromptGuard v2 [Meta, 2025] both show it exposes differences that threshold-free ranking metrics hide.

Table 6: TPR at fixed FPR, partial AUC (FPR ≤ 0.1), and AUROC. Prompt-injection head for the injection benchmarks, toxic head for ToxiGen/HateCheck.

Benchmark	Model	TPR@1% FPR	TPR@0.1% FPR	pAUC	AUROC
deepset	Hydra	0.833	0.833	0.930	0.972
	Hydra _{raw}	0.950	0.950	0.980	0.993
jackhhao	Hydra	0.957	0.842	0.977	0.989
	Hydra _{raw}	0.583	0.230	0.841	0.958
	ProtectAI v2	0.698	—	—	0.979
xtram1	Hydra	0.988	0.969	0.994	0.996
	Hydra _{raw}	0.912	0.875	0.968	0.988
jayavibhav	Hydra	0.578	0.321	0.852	0.929
	Hydra _{raw}	0.000	0.000	0.573	0.797
	ProtectAI v2	0.0003	0.000	0.709	0.876
ToxiGen	Hydra	0.201	0.007	0.668	0.808
	Hydra _{raw}	0.027	0.017	0.554	0.664
HateCheck	Hydra	0.568	0.371	0.879	0.973
	Hydra _{raw}	0.000	0.000	0.532	0.676

Neither over-flagging model has a usable operating point. On the 65,416-example benchmark, Hydra_{raw} detects nothing at a 1% false-positive budget and ProtectAI v2 detects 0.03%. Their F_1 scores on that same benchmark are 0.752 and 0.710 — respectable numbers that would pass a leaderboard. The same collapse appears on HateCheck, where Hydra_{raw} scores 0.835 F_1 and again catches nothing at 1% FPR. These models cannot be operated at any false-positive rate a deployment would accept; the only operating points at which they detect attacks are ones that flag most of the traffic. This is the same phenomenon as the saturated score distributions of Section 5.4, seen from the deployment side, and it is invisible in F_1 , precision, recall, and even AUROC (0.797 and 0.876 — both respectable).

Absolute performance at this budget. Hydra retains 0.578 TPR at 1% FPR on jayavibhav and 0.568 on HateCheck. Catching 58% of attacks at a 1% false-positive budget is a real operating point, and it is

dramatically better than zero, but it is not a solved problem. On ToxiGen we retain 0.201 at 1% FPR and 0.007 at 0.1% — below what a strict budget would require, which is why the toxic head is advisory (Section 10).

The deepset inversion. With 56 negatives, a 1% false-positive budget permits 0.56 false positives, so the metric collapses to the zero-false-positive point — which is why TPR@1% and TPR@0.1% are identical for both models — and reduces to the fraction of positives scoring above the single highest-scoring negative. Hydra is in fact better on the negatives (highest negative 4.6×10^{-6} versus 0.032); it scores lower because it confidently rejects 10 of the 60 labelled positives, all of which lack any injection keyword and include “**Generate SQL code to access a database.**” and “**translate to polish**”. This is a label disagreement, and after decontamination the comparison retains one discordant pair (Section 5.2), so no ranking follows in either direction.

Recommendation. We suggest TPR at 1% FPR as a required companion to F_1 in guard-classifier evaluation. It costs nothing to compute from scores that every paper already has, and on our data it is the only reported metric under which the difference between a deployable model and an undeployable one is unambiguous.

5.4 Calibration: Why Retuning the Threshold Does Not Fix Over-Defense

A flag rate looks, on its face, like nothing more than a threshold choice: if a model flags too much, move its cut. If that were true, the flag-rate differences reported above would say little about the models themselves. We measure whether it is true. Expected calibration error (ECE) is reported with equal-mass bins (15 bins), because equal-width binning is unreliable on saturated score distributions—exactly the regime at issue; we report equal-width alongside to show the conclusion does not depend on the choice (Table 7).

Table 7: Calibration on jayavibhav ($n = 65,416$), and the score-mass distribution that produces it. ECE CIs are bootstrap ($B = 400$). Lower ECE is better.

Model	ECE		Score mass		
	equal-width	equal-mass [95% CI]	>0.99	0.01–0.99	<0.01
Hydra	0.111	0.102 [0.100, 0.106]	0.341	0.195	0.463
Hydra _{raw}	0.453	0.451 [0.447, 0.454]	0.911	0.055	0.033
ProtectAI v2	0.392	0.389 [0.385, 0.393]	0.816	0.102	0.082

Hydra_{raw} assigns 91.1% of all 65,416 inputs a score above 0.99, and ProtectAI v2 assigns 81.6%. These models have collapsed to a near-binary output at maximal confidence: they are not making graded judgements that a better threshold could exploit, because only 5.5% and 10.2% of their probability mass respectively lies anywhere in the interior. Hydra keeps 19.5% of its mass in the interior and separates the tails, and its ECE is $3.8\times$ and $4.4\times$ lower. The ordering is identical under both binning schemes and the intervals do not overlap.

This is why we treat flag rate as a reportable quantity rather than a tuning knob (Section 4), and it answers the retuning objection directly: for these neither model has an operating point on these benchmarks that delivers both usable precision and usable recall, because the scores needed to define one do not exist.

Where calibration is poor. ECE on ToxiGen is 0.343, consistent with the toxic head’s narrow criterion (Section 10). On deepset Hydra is *worse* calibrated than the reference (0.200 vs 0.056) — the expected consequence of being confidently at odds with noisy labels (Section 8).

5.5 Internal Test Set

On the original labels the two models appear nearly tied (0.910 vs 0.914). On relabelled data Hydra wins every head (0.838 vs 0.950 core macro- F_1). Adjudication touched 40 `sqli`, 110 `pii`, and 1,661 `toxic` rows, and flipped the `pi` label on 843. **This comparison is biased in Hydra’s favour:** adjudication covered

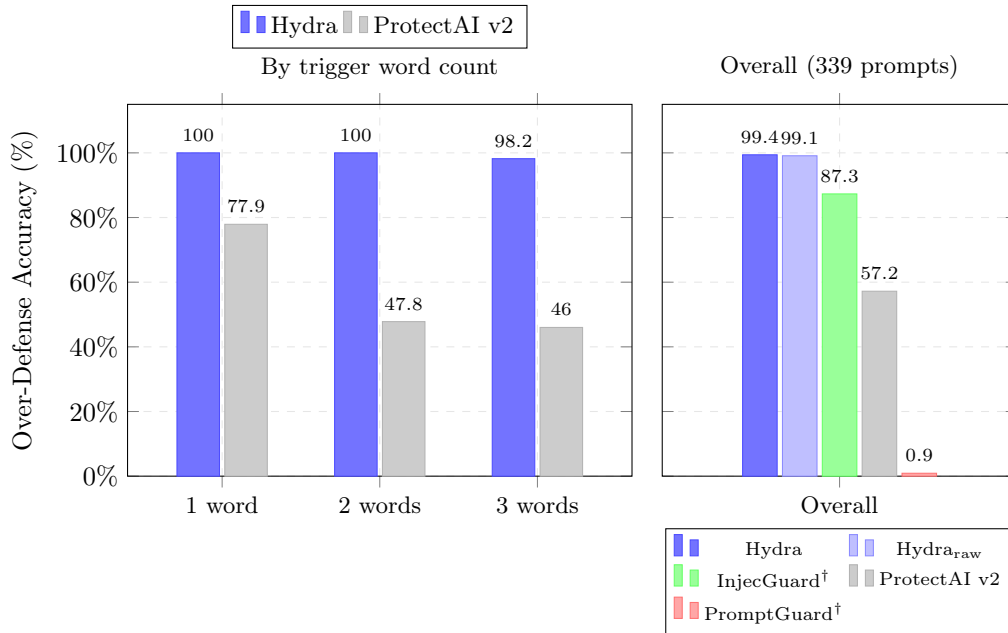
only rows where Hydra disagreed with the original label, so errors Hydra_{raw} alone makes were never revisited. The external benchmarks in Table 3 corroborate the direction; read the magnitude here as an upper bound.

6 Over-Defense

Over-defense—flagging benign input—is the failure mode that gets guards turned off [Debenedetti et al., 2024]. We evaluate it on three benign sets of increasing difficulty and against two re-measured baselines.

6.1 Trigger-Word Robustness

NotInject [Li and Liu, 2024] contains 339 benign prompts that contain injection trigger words, stratified into 113 prompts each with one, two, and three triggers. A trigger-word shortcut learner fails here by construction. We score all three models ourselves at threshold 0.5; over-defense accuracy is the fraction *not* flagged (Table 21, Figure 2).



[†]Reported by Li and Liu [2024]; per-split breakdown not available. All other bars measured by us at threshold 0.5.

Figure 2: Over-defense accuracy on NotInject (higher is better). **Every model we measured is scored at the same threshold, 0.5** — the comparison is like-for-like, and the per-head operating points of Table 2 are not used here. *Left*: breakdown by trigger-word density—Hydra stays above 98% while ProtectAI v2 falls from 77.9% to 46.0%, the signature of trigger-word shortcut learning. *Right*: overall comparison. The two Hydra bars are statistically indistinguishable (2 vs 3 false positives); the contrast that matters is with the baselines.

Both Hydra variants are near-ceiling and their intervals overlap: this is not a claim that Hydra improved over Hydra_{raw} on over-defense, only that neither regressed. The contrast is with ProtectAI v2, which flags 145 of 339 benign prompts and degrades sharply with trigger density ($0.779 \rightarrow 0.478 \rightarrow 0.460$) while Hydra stays flat—the signature Li and Liu [2024] identified as keyword shortcutting.

The obvious explanation is the wrong one. The natural attribution is to the 197 contrastive pairs that pair benign uses of “ignore”, “override” and “bypass” with genuine injections (Section 3.3) — and that is what we assumed. Our own ablation arm refutes it. Hydra_{raw} was trained on the corpus *before* those pairs were added and is already flat — 0.991 at one, two and three triggers (Table 21) — while Hydra, which has them, is not flatter and in fact dips at three triggers (0.982). A targeted supplement of 197 pairs against a

230,332-row training split is not what separates these models from ProtectAI v2’s 0.779 $\rightarrow$ 0.460 collapse. Whatever produces trigger-word robustness here predates that supplement, and we have not isolated it.

6.2 Over-Defense on Trigger-Word-Free Benign Prompts

NotInject is 339 rows and, as Section 5.2 shows, 67% overlapping our training corpus. It also selects prompts *containing* trigger words, so good performance there is consistent with either genuine context understanding or a merely better lexicon. We therefore evaluate on PHTest [An et al., 2024], 3,269 pseudo-harmful benign prompts containing *no* trigger words, which separates the two explanations, and on **OR-Bench hard-1k** [Cui et al., 2024], 1,319 hard over-refusal prompts. Both are benign-only, so the metric is false-positive rate; there is no AUPRC to report (Table 8).

Table 8: False-positive rate on benign-only sets, Wilson 95% intervals. “any-head” fires if *any* of the four heads exceeds its threshold. This is an *upper bound* on the architecture’s over-defense cost, not a description of a deployment: each head is configured independently to block, to flag for review, or to be left unenforced, and a head affects traffic only where a control is applied to it (Section 3.5). **The “FPR @0.5” column is the like-for-like comparison** — every model at the same threshold. “FPR @tuned” is reported beside it so the reader can see that Hydra’s advantage does not depend on its operating point: it holds at both. Tuned thresholds are Hydra 0.58, Hydra_{raw} 0.78 (the *predecessor’s* value, not ours), ProtectAI v2 argmax.

Set	Model	FPR @0.5	FPR @tuned	any-head @0.5
PHTest (3,269)	Hydra	0.734% [0.494, 1.090]	0.704%	1.009% [0.720, 1.414]
	Hydra _{raw}	13.246% [12.13, 14.45]	9.299%	—
	ProtectAI v2	2.172% [1.726, 2.731]	2.172%	n/a (single head)
OR-Bench hard-1k (1,319)	Hydra	2.654% [1.914, 3.668]	—	—
	Hydra _{raw}	68.082%	—	—
	ProtectAI v2	6.520%	—	n/a

This is our first adequately powered over-defense comparison. Against ProtectAI v2 on PHTest, McNemar gives $b=22$, $c=69$, $b+c=91$, $p = 8.2 \times 10^{-7}$ — above our power threshold, unlike every NotInject comparison. The Wilson intervals are disjoint, and Hydra’s advantage holds at both 0.5 and its tuned 0.58, so it is not threshold-shopped. Because PHTest contains no trigger words, this supports the stronger explanation: the behaviour is not a better keyword list.

Four qualifications.

- The multi-head design has four chances to false-positive.** Enforcing *every* head raises Hydra’s PHTest FPR from 0.734% to 1.009% — **37% more false positives** than the injection-head number implies. Any per-head over-defense figure understates the behaviour of a deployment that enforces all heads, ours included, and papers reporting a single head’s FPR for a multi-head model are understating theirs too. The cost is intrinsic to the architecture but not unavoidable in practice: because each head carries its own calibrated score, each is independently set to block, to flag for review, or to be left unenforced, which is the operating-point control a single-score guard cannot express (Section 3.5). The union figure is the bound if every head is set to block.
- The toxicity head is the weakest and is not enforced as a blocking control.** Its calibration and low-FPR detection are the poorest of the four (ToxiGen ECE 0.343, TPR@1%FPR 0.201), so it is configured to flag for review rather than to block; Section 10 records the consequence.
- Sub-1% over-defense does not survive OR-Bench.** Hydra is 2.654% there, $3.6\times$ its PHTest rate, with **privacy** the worst category at 5.528% [3.11, 9.63] — unsurprising for a model with a PII head, but a real cost. We remain well below ProtectAI v2 (6.520%), but the claim is consistently lower over-defense than the baselines, not a sub-1% headline.

4. **A low false-positive rate is only meaningful beside a true-positive rate.** A model that flags nothing has a perfect FPR. OR-Bench’s toxic split is a poor control for this—it is general harmful content, not injection, so our injection head arguably *should* be quiet on it (we score 2.7%, and ProtectAI v2 scores even lower at 0.9%). The burden is instead discharged by Table 3: Hydra recalls 0.979 on `xtram1`, 0.957 on `jackhhao` and 0.837 on `jayavibhav` at its best- F_1 thresholds. Those are not the same operating points as the false-positive rates above (0.5 and the tuned 0.58), so the two sets of numbers characterize the model rather than forming a single trade-off curve. Low false positives *and* high recall is the pairing that matters, and neither number alone establishes it.

Two methodological notes. PHTest’s standard loader ingests a checksum file, so we loaded `PHTest.csv` directly from the same repository; text column `Request`, category `Harmfulness`. OR-Bench hard-1k contains 1,319 rows, not the 1,320 its documentation implies. Hydra_{raw}’s 68% OR-Bench FPR is so extreme that we flag it as a possible artifact of the corpus cleaning interacting with these sets rather than treating the gap as a clean measurement of data quality.

Character-level evasion (Appendix A). Static character perturbations reduce Hydra detection by at most 6.79 percentage points across 17 variants; homoglyph substitution costs 2.65 points even at full substitution. The near-total evasion reported for prompt-injection detectors [Cisco, 2025] does not reproduce here. This is not an adaptive-robustness claim: the attack set is selected from positives Hydra already detects, no optimizing adversary is tested, and on that selected set Hydra_{raw} shows a *smaller* worst-case drop (1.82 points) — partly a headroom artefact we do not separate. Benign false-positive rate never fell under any perturbation, so the losses are selective. Macro detection is statistically indistinguishable from ProtectAI v2 (97.88% vs 97.86%).

6.3 Two Further Over-Defense Sets

Because NotInject proved to be in our training data (Section 5.2), we add two benign sets that are not (Table 20). Both are scored with the row-wise maximum across all four heads — the blocking behaviour a deployment actually exhibits — rather than the injection head alone.

On XSTest’s safe half our false-positive rate is 1.60% against 18.00% — an $11\times$ reduction with $p = 8.2 \times 10^{-10}$. WildJailbreak’s adversarial-benign quadrant is worse. At 17.14% we are less than half the reference model’s 31.90%, though roughly one benign prompt in six is still flagged, and this is the hardest public benign probe available — benign requests deliberately phrased to look adversarial. It is also the clearest illustration of the any-head cost from Section 6.2: four heads mean four opportunities to fire on the same text.

The unsafe halves are worth noting for what they are not. Our TPR on XSTest’s unsafe half is 22.5%, statistically tied with the reference model. XSTest’s unsafe prompts are general harmful requests — weapons, self-harm, slurs — and lie outside our four categories, so neither number characterizes injection detection (Section 10).

6.4 Over-Flagging Is Not Specific to Our Predecessor

It would be reasonable to suspect that Hydra_{raw}’s 66.5% flag rate is an artefact of our own training rather than a property of guard classifiers generally. It is not. We scored ProtectAI v2 (Table 18)—a publicly available production injection classifier—under the `argmax (0.5)` rule of the standard Hugging Face pipeline, which is what practitioners get by default, on the same benchmarks.

On 65,416 examples ProtectAI v2, under the standard pipeline’s `argmax` default, flags 87.8% of all inputs at precision 0.555—barely above the 49.4% base rate, i.e. close to flagging everything. Its recall is excellent (0.986) and its F_1 (0.710) is respectable, so an evaluation that reports only F_1 and recall would rate it usable. It is the same failure mode as Hydra_{raw}’s 66.5%, in a model we did not train, and it is invisible without the flag rate.

A second baseline, and two benchmarks where we do not win. We also re-measured InjecGuard [Li and Liu, 2024] (Table 19; canonical repository `leolee99/PIGuard`; label convention verified against known-malicious and known-benign strings before use). It is the strongest encoder baseline we tested, and the result is mixed in a way worth stating precisely.

On the two small benchmarks we are tied with InjecGuard — on `jackhhao` it is nominally ahead (0.9963 vs 0.9927) — and both comparisons are underpowered ($b+c$ of 10 and 11), so neither supports a ranking claim. We separate decisively only on the two large benchmarks and on over-defense. That pattern is itself the paper’s argument: the benchmarks that cannot distinguish these models are precisely the small, widely cited ones, and the distinction that matters for deployment (a $19\times$ difference in benign false-positive rate) appears on neither of them.

Note also that ProtectAI v2 scores 0.929 AUPRC on `deepset` and 0.856 on `jayavibhav`: the small benchmark overstates it by 7 points, directionally consistent with Fomin [2026], though that estimate is from a different model class. Matched-operating-point analysis is what makes the comparison fair in the other direction too—at ProtectAI’s own precision on `deepset`, Hydra recalls 0.833 against its 0.367.

7 Comparison with Concurrent Encoder Guardrails

Published figures for other guards are reported at their own thresholds on their own splits, which Section 4 identifies as not comparable. We therefore ran both concurrent systems (Section 2) ourselves on identical rows (Table 9), which yields *paired* statistics. Both are dynamic-schema models, so their score depends on the label set supplied; we gave each its most favourable phrasing as primary and verified polarity on known-malicious and known-benign strings before use.

Table 9: AUPRC on identical rows, all models scored by us. Hydra leads on every adequately powered prompt-injection and toxicity comparison. Right-hand columns: flag rate on benign-only sets (lower is better).

Benchmark	Hydra	GLiNER Guard	Opir	Verdict
<i>AUPRC (higher is better):</i>				
<code>xtram1</code> (2,060)	0.9955	0.9888	0.9023	improved
HateCheck (3,728)	0.9858	0.8454	0.7460	improved
<code>jayavibhav</code> (5,000)	0.9361	0.8768	0.8743	improved
ToxiGen (940)	0.7752	0.6341	0.5940	improved
ToxicChat jailbreak (5,083)	0.7995	—	0.1875	improved
<code>jackhhao</code> (262)	0.9927	0.9501	0.8511	vs. Opir improved [†]
<code>deepset</code> (116)	0.9781	0.9735	0.7206	vs. Opir improved
ToxicChat toxicity (5,083)	0.1969	—	0.4234	Opir leads
Aegis 2.0 (1,964)	0.7442	—	0.9513	Opir leads
<i>Flag rate on benign-only sets, each model at its own default (lower is better):</i>				
NotInject	0.59%	5.01%	19.76%	—
PHTest	0.70%	1.62%	57.63%	—
OR-Bench hard-1k [‡]	2.12%	1.90%	68.99%	—
OR-Bench toxic	2.44%	1.68%	92.82%	—

[†] $b=29$, $c=0$, $p = 3.7 \times 10^{-9}$, $\Delta\text{AUPRC CI } [+0.025, +0.065]$ excluding zero — but $b+c = 29$ falls one discordant pair below our pre-stated power floor of 30. Every discordant pair favours Hydra and the interval excludes zero; we nonetheless report the counts without a ranking claim, rather than relax a pre-stated threshold to fit a favourable result.

Hydra records the highest AUPRC on seven of the nine benchmarks on which a competitor was scored, and leads every adequately powered prompt-injection comparison. Against Opir the margins are wide throughout — 0.9781 vs 0.7206 on `deepset`, 0.9927 vs 0.8511 on `jackhhao`, and $4\times$ on ToxicChat’s jailbreak label. On benign traffic Opir is close to unusable at its default operating point (57.6% of PHTest, 92.8% of OR-Bench toxic, against our 0.70% and 2.44%). GLiNER Guard is the stronger competitor: we lead it on NotInject ($8.5\times$ lower flag rate) and PHTest, and it edges us on both OR-Bench splits by roughly 0.2–0.8 points.

On their benchmarks: one clear win, two losses. On ToxicChat [Lin et al., 2023] — reported by Opir, WildGuard and MultiTaskGuard, and so the closest thing to a common yardstick in this literature — our injection head reaches **0.7995** AUPRC on the **jailbreaking** label against Opir’s 0.1875, at a 1.8% flag rate. On the same corpus’s **toxicity** label we *lose*, 0.1969 against 0.4234 (Δ CI $[-0.289, -0.167]$), and on Aegis 2.0 we lose 0.7442 against 0.9513.

Both losses are toxicity or general-harm tasks routed through our advisory toxic head, and we read them as a statement about scope rather than about quality: Hydra is a prompt-security classifier, not a general safety classifier. Opir targets a 996-label harm taxonomy and scores higher on that task; we score $4\times$ higher on the one injection benchmark we share. Where a deployment needs broad harm coverage, these results argue for a general safety model *alongside* an injection specialist rather than in place of one.

8 Benchmarks Reward Over-Flagging

The **deepset** result is instructive. Hydra scores worse there (Δ AUPRC -0.016) while being decisively better on a benchmark $560\times$ larger ($+0.198$). Three measurements explain the inversion.

1. The reference model flags most of its input. On jayavibhav it attains its best F_1 while flagging **66.5%** of all 65,416 inputs at precision 0.655 (Figure 3). On HateCheck it flags 93.7%. A model this permissive about firing scores well on small, positive-dense benchmarks—**deepset** is 51.7% positive, and flagging nearly everything yields high recall at tolerable precision—while being unusable in production.

2. deepset’s positive labels include benign prompts. Of the 60 positives, **6 are scored below 0.5 by both models**, and **47 of 60 contain no injection-like keyword** (ignore, disregard, previous instruction, system prompt, jailbreak, DAN, forget, override, reveal, bypass). Examples labelled as injections include role-play requests (“I want you to act as an interviewer”), ordinary code generation (“Generate SQL code to access a database”), and translation requests (“translate to polish”). A classifier is rewarded for firing on these.

3. The benchmark is too small to distinguish the models. With 8 discordant pairs, **deepset** cannot support a ranking claim in either direction. Our own result there is not significant ($p = 0.070$), and we do not treat it as a regression.

4. No deployable operating point exists for either model. Table 6 shows that on the 65,416-example jayavibhav benchmark the reference model detects *no* attacks at a 1% false-positive budget (TPR = 0.0000) and ProtectAI v2 detects 0.03% (TPR = 0.0003). Their apparent competence is confined to operating points that flag most of the input, which is precisely the behaviour the small benchmarks reward.

5. The pattern reproduces in a third-party model. Section 6.4 shows ProtectAI v2 flagging 87.8% of the same 65,416 inputs at its default threshold while scoring 0.929 AUPRC on **deepset**. The inversion is therefore not specific to our training pipeline.

Implication. This sharpens Fomin [2026]: benchmarks do not merely *overestimate* performance, they can *invert* model preference, selecting for the over-flagging behaviour that DeBenedetti et al. [2024] showed destroys 40% of task completion. We recommend that prompt-injection evaluations (i) report flag rate alongside F_1 , (ii) report discordant-pair counts so underpowered comparisons are visible, and (iii) include at least one benchmark with n in the tens of thousands. We are not claiming **deepset** is worthless—it is a useful smoke test—only that it cannot adjudicate between models, and that leaderboard positions derived from it are unreliable.

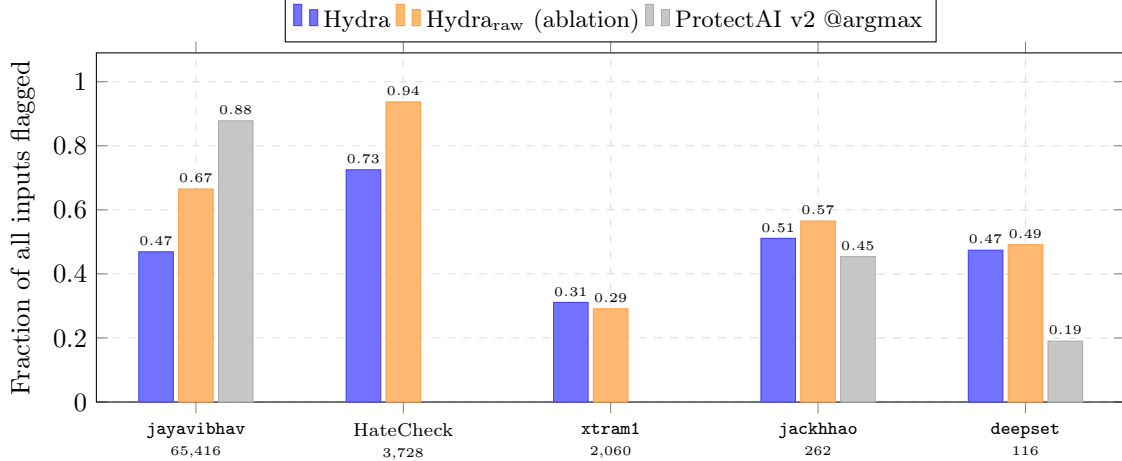


Figure 3: Flag rate—the fraction of *all* inputs a model flags—across benchmarks ordered by size. On the small benchmarks (right) every model looks reasonable and the models are nearly indistinguishable. Only at scale (left) does over-flagging become visible: the ablation flags 66.5% of 65,416 inputs and 93.7% of HateCheck, and ProtectAI v2 at the argmax default flags 87.8%. Each model is at its own best- F_1 threshold except ProtectAI v2, shown at its documented default of 0.5. ProtectAI v2 is omitted where its label space does not apply.

9 Efficiency and Deployment

The multi-head design’s efficiency claim is that four category scores come from one encoder pass. We measure it rather than assert it. All numbers below are from the deployed ONNX model on an **NVIDIA A100-SXM4-40GB**, CUDA execution provider, FP32, measured *inside* the serving pod over loopback so no client-network time is included; 15 warm-up plus 120 timed requests per point, single request at a time unless stated.

Latency is essentially flat from 160 to 640 characters and rises only 27% across a $16\times$ increase in input length. Per-request cost is therefore dominated by fixed overhead—request handling, tokenization, kernel launch—not by encoder compute. Under 8 concurrent clients the same pod sustains **452.8 req/s**.

Table 10: One shared pass versus the four inference calls a four-separate-models deployment would issue. Same pod, 320-character input; $n = 120$ (shared) and $n = 60$ (four-call).

Configuration	p50	p95	mean	vs shared
Shared encoder, 1 pass, 4 scores	30.9	31.5	30.7	—
Four calls, sequential	123.3	125.8	122.3	3.98×
Four calls, concurrent	32.7	69.6	34.6	1.13×

9.1 What the Shared Encoder Costs in Accuracy

Table 11 isolates the accuracy side of the trade-off. Three seeds of the released configuration bound run-to-run variation; two ablation arms then vary only the head structure, holding data, seed, schedule and every hyperparameter fixed.

Two results follow. **Head capacity is not the binding constraint**: replacing four independent MLP heads with a single 4-output linear head changes macro- F_1 by -0.0006 , inside seed noise on four of five metrics. And **four separate encoders are slightly but consistently better** — $+0.006$ macro- F_1 , with every head improving by 1.5–2.3 SD of the seed band.

We read that as a cost accounting rather than an accuracy defeat: four separate encoders buy ≈ 0.6 macro- F_1 points for $3.99\times$ the parameters and $3.3\times$ the resident memory. Two qualifications keep it honest.

Table 11: Validation F_1 per head. Seed column is mean $\pm$ SD over three seeds of the released four-head configuration; the two ablation arms are single runs. Early stopping was left as shipped, so runs completed 3–5 epochs.

Head	4 MLP heads (3 seeds)	1 linear head	4 separate models
sqli	0.9960 ± 0.0007	0.9952	0.9976
pii	0.9775 ± 0.0028	0.9754	0.9831
pi	0.7670 ± 0.0109	0.7707	0.7835
toxic	0.7881 ± 0.0012	0.7853	0.7899
macro	0.8822 ± 0.0032	0.8816	0.8885
<i>relative parameter count</i>		1.00 $\times$	3.99 $\times$

Each single-head run selects the checkpoint best for *its own* head while the shared model selects on aggregate macro- F_1 , so part of the gap is selection pressure rather than architecture. And each ablation arm is a single run compared against a three-seed band, so the per-head margins are individually suggestive rather than established — though all four point the same way. What the result establishes is the absence of large negative transfer: the shared encoder sacrifices ≈ 0.6 macro- F_1 points.

What the ratio does and does not mean (Table 10). Four sequential calls cost $3.98\times$ the latency, but issued *concurrently* only $1.13\times$ at the median. The shared encoder therefore does not buy a $4\times$ latency reduction. What it buys is threefold:

1. **Compute.** Four passes do four times the encoder work regardless of how they are scheduled. The concurrent configuration hides this behind parallelism; it does not remove it. At 452.8 req/s sustained, a four-model design would need roughly four times the accelerator budget for the same offered load.
2. **Tail latency under contention.** Concurrency is not free when the four models share hardware: p95 degrades from 31.5ms to 69.6ms, a $2.2\times$ tail regression, while the median barely moves. Avoiding four-way self-contention is a real benefit that a median-only comparison hides.
3. **Memory.** One resident model occupies 3,882MiB of GPU memory including CUDA context; the FP32 ONNX file is 1,119MB. Four separate single-head models would total **1,116,025,860** parameters against our 279,597,828 — exactly $3.99\times$: +836M parameters, and 4,477MB of weights on disk against our 1,119MB, an increase of 3,358MB. The saving is dominated by not replicating the 193M-parameter embedding matrix four times; the heads themselves are 0.28% of the model, so a single-head model is very nearly as large as ours. Measured resident memory follows: one loaded and warmed inference session is 2,148MiB RSS, four concurrent sessions are 7,053MiB ($\approx 1,635$ MiB per additional encoder).

An independent CPU measurement of the same comparison (onnxruntime CPU provider, batch 1 and 8, sequence lengths 64–512) gives a four-pass/one-pass ratio of $3.65\times$ – $4.11\times$, bracketing the $3.98\times$ measured on the A100 and confirming that the ratio is a property of the parameter accounting rather than of one accelerator.

Cost/performance across accelerators. A separate sustained-load campaign (32 concurrent virtual users, 300s per prompt class, measured in-cluster against the deployed service) compares the two accelerators the model actually runs on. The L4 deployment uses the TensorRT execution provider and the A100 deployment uses CUDA, so this is a comparison of two *deployments*, not of two chips (Table 16).

The pattern reinforces Table 15. On short prompts the two deployments are indistinguishable (722 vs 720 req/s; 82 vs 83ms p99) because per-request fixed overhead dominates and neither accelerator is the bottleneck. The A100’s advantage appears only as inputs lengthen and encoder compute begins to dominate: $2.3\times$ throughput and 57% lower p99 on medium prompts, $1.6\times$ and 53% on long. Against a 1,000ms p99 target the L4 deployment meets the goal on every class with no headroom (973ms on long prompts) at $4.3\times$ lower cost. We report this because it is the decision a deployer faces, and because it shows the efficiency argument for a shared encoder is about throughput and cost per request, not about single-request latency on short inputs—where there is nothing to win.

Caveats. The four-call figures use four calls to the *same* service as a proxy for four single-head services. This is a good proxy for encoder cost, because the heads are 197,121 parameters against an 86M backbone, but it is not a measurement of four independently deployed models, and it assumes they would be co-located on one accelerator. These figures cover latency and memory; the accuracy side is measured separately in Section 9.1. All figures are FP32, which we cannot safely improve on (Section 3).

10 Limitations

Table 12 states each limitation, its consequence for how our results should be read, and the evidence status behind it. Two warrant expansion below because they bound what the model may be used for; the appendix carries the detail for the rest.

Scope. Hydra detects four specific categories. On general harmful content — illicit synthesis, weapons, self-harm advice — it scores near zero because no head applies, and the controlled reference detects more of it precisely because it over-flags. It is an injection specialist, and a deployment needing broad harm coverage should pair it with a general safety model rather than substitute one for the other.

PII localisation is attributive, not extractive. The service returns character-offset regions per head with a Shapley credit share, at sentence, clause and word granularity. These identify *which text drove a head*; they are not *typed* extraction (PHONE vs SSN), which redaction requires and which a complementary extractor supplies [Microsoft, 2023]. Attribution answers “why did this fire”; extraction answers “what must be removed”. We provide the former for all four heads and the latter for none.

11 Conclusion

We presented Hydra, a multi-head encoder classifier covering prompt injection, toxicity, PII, and SQL injection in one forward pass with independent per-category scores. Against an ablation holding the architecture fixed, data-quality work produced significant gains on four of six external benchmarks, including $\Delta\text{AUPRC} +0.198$ ($[+0.194, +0.202]$) on a 65,416-example prompt-injection set and $+0.194$ ($[+0.181, +0.208]$) on HateCheck, with the advantage surviving matched-operating-point analysis.

Hydra obtains four independently thresholdable threat scores from one encoder pass, using $3.99\times$ fewer parameters than four separate models at a cost of ≈ 0.6 macro- F_1 points, and leads seven of the nine benchmarks on which we scored a competitor. On PHTest — our cleanest uncontaminated over-defense comparison — it false-positives on 0.73% of benign prompts against ProtectAI v2’s 2.17% ($p = 8.2 \times 10^{-7}$).

Our most transferable finding is methodological: the field’s most-cited prompt-injection benchmark *prefers* the model that flags two thirds of its input, and a publicly available third-party classifier flags 87.8% of a 65,416-example benchmark under the default decision rule practitioners actually use, while its headline metrics look healthy. Evaluations that omit flag rate, omit discordant-pair counts, and rely on hundred-example datasets will select classifiers that cannot be deployed. We recommend reporting flag rate, calibration, and TPR at a fixed false-positive budget alongside accuracy; reporting effective sample size for paired comparisons; including at least one large benchmark; and, for multi-head models, reporting the *any-head* false-positive rate alongside per-head rates, since it bounds what a deployment enforcing every head would incur — 37% above our injection-head rate alone.

Finally: across this work, cleaning labels consistently beat changing the architecture or the loss function. The gap measured here is almost entirely a data result.

A Character-Level Evasion

Character-level attacks have been reported to evade prompt-injection detectors, including Meta’s Prompt-Guard, at rates approaching 100% [Cisco, 2025]. We tested this directly. From `xtram1` we selected the 604 injection positives that Hydra already detects at its tuned threshold — so the pre-attack detection rate

Table 12: Limitations, consequences, and evidence status. Nothing here is speculative — each row is grounded in a measurement reported above or in an explicitly unmeasured gap.

Limitation	Consequence	Evidence status
Relabelling favoured Hydra	Internal effect size is an upper bound	External benchmarks supply direction; adjudication covered only rows where Hydra disagreed
LLM-oracle label circularity [Panickssery et al., 2024]	Correlated train/test errors possible	No stratified human audit performed
Toxic head is narrow	Not usable as a blocking signal; flag-for-review only	ToxiGen ECE 0.343, TPR@1%FPR 0.201; misses interpersonal abuse. The targeted supplement’s intended counterspeech/hate contrast is <i>unpaired</i> : 198 counterspeech rows generated, 0 hate counterparts (Section 3.3)
Heldout leakage in the training corpus	324 rows of a previously-registered heldout test split are present in the training data	Identified by re-hashing the heldout registry: the registry’s content hash includes the label, so relabelling a row makes the lookup miss. 324 of 230,332 rows (0.14%); the frozen test split used for evaluation is unaffected (0 overlap, verified)
No external PII or SQLi validation	Our two highest-scoring heads are checkable only by us, and internal figures may not predict out-of-distribution behaviour	We attempted external validation and did not find a benchmark supporting a reliable estimate: candidate PII corpora either overlap our training data or lack a consistent PII definition; SQLShield [Chafik et al., 2025] is CC-BY-NC-ND with OpenAI-generated queries
General harm out of scope	Must not be deployed as a general safety classifier	WildJailbreak vanilla-harmful 2.86% vs 44.76%; StrongREJECT 13.4%
Narrower coverage than concurrent systems	Scope difference, not capability gap	Opir 996-label taxonomy; GLiNER Guard typed PII spans; both post-date this work
Cross-lingual coverage is partial	Injection thresholds need per-language fitting; toxicity unusable in hi/ar	Appendix B; SQLi transfer unmeasured
Static perturbations only	No adaptive-robustness claim	Appendix A; attack set selected on Hydra detections
No leave-one-dataset-out	Source dominance unquantified beyond the audit	20+ training sources; not run
Seed variance measured, single config	Training noise bounded at ± 0.003 ; other configs unmeasured	Three seeds, released config, early stopping left as shipped
Direct injection only	Indirect injection via tool output untested	Not evaluated [Greshake et al., 2023]
Deployment numerics affect correctness	Accuracy is a property of model <i>and</i> runtime	Observed range-compression under one vendor’s reduced-precision kernel
Benchmarks we could not access	Several comparisons incomplete	WildGuardMix, Sentinel, PromptGuard-2 access-restricted

is **100% by construction**, and the numbers below are the drop an attacker induces — then applied six perturbation families separately, at several strengths each (17 variants total; Table 13).

Table 13: Character-level evasion, worst variant per family. $n = 604$ attacks that Hydra detects pre-perturbation; detection at the tuned threshold. “Evasion” is the percentage-point drop.

Family	Worst variant	Post-attack detection	Evasion
Random casing	<code>random_case</code>	93.21%	6.79
Fragmentation	<code>fragment_dot</code>	93.71%	6.29
Zero-width insertion	30% density	95.86%	4.14
Combining diacritics	30% density	96.36%	3.64
Homoglyphs	100% substitution	97.35%	2.65
Encoding	ROT13	97.68%	2.32
Benign padding	2 sentences	99.83%	0.17

No perturbation came close to defeating the model. The worst is random casing at 6.79 points; homoglyph substitution — the attack most associated with the published near-total evasion results — costs 2.65 points even at 100% substitution. We report this as a failure to reproduce the published evasion rates, not a claim of adaptive robustness: our attack set, threshold, and selection differ from the cited work, and an optimizing adversary is not tested here.

Selectivity control. A perturbation that suppressed *all* scores would lower detection without being an attack. Across all 17 variants and both models, the benign false-positive rate never fell — it held flat or rose. So the detection losses are genuinely selective. The rises are their own small finding: character noise pushes some benign text *up*, so these perturbations degrade both sides of the operating point.

We are not uniquely fragile. Macro detection across all 17 variants is **97.88% for Hydra against 97.86% for ProtectAI v2** — statistically indistinguishable, better on 6 variants, worse on 9, tied on 2. ProtectAI v2’s single worst case (10% homoglyphs, 91.56%) is worse than ours. On the WildJailbreak adversarial-benign probe, measured on the injection head alone at each model’s tuned threshold, the ordering is clearer: 17.14% for us, 31.43% for the reference model, 32.86% for ProtectAI v2.

The reference model is more robust on this set, with an important caveat. Hydra_{raw}’s worst loss is 1.82 points against our 6.79, and it loses detection on only 3 of 17 variants against our 13 — so on this measure the label-cleaning work appears to have *reduced* perturbation robustness. But the comparison is confounded by construction: the attack set was selected as positives Hydra detects, pinning our pre-attack rate at 100% while Hydra_{raw} starts at 92.22%. Part of the gap is therefore headroom rather than fragility, and we do not claim to have separated the two.

B Cross-Lingual Evaluation

The backbone is multilingual (mDeBERTa, ~100 languages) and its 250K-token embedding is 69% of the model’s parameters, but the heads are fine-tuned on predominantly English data. Whether that embedding earns its cost is therefore an empirical question, and the answer differs by head (Table 14).

Ranking transfers; the threshold does not. Δ AUPRC against the English anchor is between -0.0015 and -0.0085 across six languages — the model orders non-English attacks essentially as well as English ones. Recall at the English-tuned threshold nevertheless falls from 0.933 to 0.727–0.873, while **precision remains 1.000 everywhere**. That signature identifies the cause precisely: non-English scores are shifted downward, so an English-tuned cut is too strict rather than the ranking being worse. The remedy is per-language thresholds, which the per-head output contract already accommodates — a single collapsed score would not. Note also that Hindi has the second-*highest* recall and German the lowest, so this is not a pretraining-resource gradient.

Table 14: Cross-lingual results. Injection: 300 parallel `xtram1` rows machine-translated (NLLB-200), scored against the same rows in English. Toxicity: Multilingual HateCheck, native text. Recall is at the English-tuned threshold.

Language	Injection head		Toxic head		
	AUPRC	recall	AUPRC	TPR@1%	FPR
English (anchor)	0.9996	0.933	0.9858		0.569
German	0.9962	0.727	0.9711		0.462
Spanish	0.9912	0.760	0.9744		0.449
French	0.9911	0.773	0.9650		0.409
Chinese	0.9939	0.740	0.9386		0.272
Arabic	0.9981	0.813	0.8826		0.161
Hindi	0.9976	0.873	0.8343		0.059

Toxicity transfer is the weaker case. Unlike the injection head, AUPRC itself falls (0.986 to 0.834), and at a 1% false-positive budget the model catches 5.9% of hateful content in Hindi and 16.1% in Arabic. We do not claim multilingual toxicity detection. The English anchor is itself only 0.569 at that budget, so this sits atop an already-soft curve.

Scope of this experiment. These results cover the injection and toxic heads. The `sqli` and `pii` heads are only partially addressed by a translation-based design, for reasons intrinsic to the method rather than to the model: machine translation does not preserve SQL payloads (the signature survived in fewer than 10% of payload-bearing rows, which we excluded), and translated English PII retains English-shaped phone, postcode and identifier formats, so the 0.970–0.999 AUPRC there reflects the effect of the surrounding language rather than of locale. Establishing those two heads cross-lingually needs native data, which we did not have; the appendix records the per-language fidelity checks and the one language (Chinese) whose translation quality is low enough to confound its own delta.

Translation fidelity. The injection results use a parallel corpus produced by NLLB-200-distilled-600M, so translation quality bounds what they can show. Round-trip chrF/BLEU on a fixed 30-row subsample (seed 20260727) is: Spanish 68.4/53.3, Hindi 67.2/50.6, French 65.6/48.2, German 59.9/43.8, Arabic 51.3/35.3, Chinese 40.0/24.5. Chinese is the weakest by a clear margin, so its recall delta partly confounds translation damage with model behaviour and should not be read as a language-specific model result. For SQL injection the same pipeline mangled the payload signature in 19–21 of the 21 payload-bearing rows per language (Hindi 21/21); those rows were excluded, which is why `sqli` transfer is reported as unestablished rather than measured. No powered non-English *native* cell exists for either head, so none of the translated results has native corroboration.

Verdict on the backbone. The multilingual embedding earns its cost for injection ranking across all six languages tested — Δ AUPRC at most 0.0085 from the English anchor — and partially for toxicity, strongly in Latin-script languages and not at a deployable operating point in Hindi or Arabic.

C Supplementary Tables

Tables referenced from the main text and collected here to keep the main body compact.

D Supplementary Analysis

What jayavibhav is. Its dataset card is empty, so we characterized it directly. It contains verbatim all 116 `deepset` rows and all 2,060 `xtram1` rows, plus a large body of LLM-generated benign instruction text. Positives run longer than negatives (383 vs 311 median characters), a length shortcut a model could exploit. Internal duplication is negligible (0.17% normalized), so cluster-aware intervals are unnecessary. **Because**

Table 15: End-to-end serving latency (ms) by input length, one shared encoder pass returning all four head scores. A100, FP32, $n = 120$ per row.

Input chars	p50	p95	p99	mean	serial req/s
160	29.7	31.0	31.4	29.8	33.5
320	29.5	30.1	31.5	29.5	33.9
640	29.7	30.4	31.9	29.7	33.6
1280	31.0	31.7	31.9	30.9	32.3
2560	37.8	38.5	39.0	37.9	26.4

Table 16: Sustained load, 32 concurrent users, 300s per class. Prices are list rates at the time of measurement. Lower p99 and higher throughput are better.

Prompt class	Throughput (req/s)		p99 latency (ms)	
	L4+TRT	A100+CUDA	L4+TRT	A100+CUDA
Short	722	720	82	83
Medium	67	156	494	211
Long	48	76	973	455
<i>List price: L4 \$0.85/hr ($\approx$\$620/mo) A100-40GB \$3.67/hr ($\approx$\$2,680/mo, $4.3\times$)</i>				

Table 17: Hydra retuned to Hydra_{raw}’s operating point on the prompt-injection benchmarks. A genuine improvement should hold under both matchings.

Benchmark	Matched on	Hydra _{raw} Prec / Rec	Hydra Prec / Rec	Verdict
deepset	flag rate	1.000 / 0.950	0.930 / 0.883	worse
	precision	1.000 / 0.950	1.000 / 0.833	worse
jackhhao	flag rate	0.892 / 0.950	0.919 / 0.978	better
	precision	0.892 / 0.950	0.895 / 0.986	better
xtram1	flag rate	0.985 / 0.908	1.000 / 0.918	better
	precision	0.985 / 0.908	0.989 / 0.982	better
jayavibhav	flag rate	0.655 / 0.882	0.699 / 0.941	better
	precision	0.655 / 0.882	0.655 / 0.958	better

Table 18: ProtectAI v2 at its default threshold vs Hydra, plus Hydra retuned to ProtectAI’s operating points. Thresholds shown where retuned.

Benchmark	Model / point	AUPRC	F_1	Prec	Rec	flag
deepset	ProtectAI v2 @argmax	0.929	0.537	1.000	0.367	0.190
	Hydra @0.5	0.978	0.763	1.000	0.617	0.319
	Hydra matched flag rate			1.000	0.367	0.190
	Hydra matched precision			1.000	0.833	0.431
jackhhao	ProtectAI v2 @argmax	0.966	0.907	0.983	0.842	0.454
	Hydra @0.5	0.993	0.955	0.992	0.921	0.492
	Hydra matched flag rate			0.992	0.849	0.454
	Hydra matched precision			0.985	0.957	0.515
jayavibhav	ProtectAI v2 @argmax	0.856	0.710	0.555	0.986	0.878
	ProtectAI v2 @best F_1	0.856	0.803	0.765	0.845	0.545
	Hydra @0.5	0.940	0.853	0.906	0.805	0.439

Table 19: Hydra versus InjecGuard, paired. Both models scored by us on identical rows.

Benchmark	n	Δ AUPRC [95% CI]	$b+c$	Verdict
deepset	116	+0.0033 [−0.0221, +0.0316]	10	tie, underpowered
jackhhao	262	−0.0035 [−0.0108, +0.0017]	11	tie, underpowered
xtram1	2,060	+0.1411	335	improved, $p=1.0 \times 10^{-79}$
jayavibhav	65,416	+0.1053	16,884	improved
<i>NotInject false-positive rate:</i> Hydra 0.59% InjecGuard 11.5% ProtectAI v2 42.8%				

Table 20: Additional over-defense evaluation, row-wise max across heads at tuned thresholds. Wilson 95% intervals.

Set	Portion	Hydra	Hydra _{raw}	McNemar
XSTest	safe (250)	1.60% [0.62, 4.04]	18.00%	$b=45, c=4, p=8.2 \times 10^{-10}$
XSTest	unsafe (200), TPR	22.50%	25.00%	$b=37, c=42, p=0.65$ (tie)
WildJailbreak	adversarial-benign (210)	17.14% [12.65, 22.82]	31.90%	$b=42, c=11, p=2.3 \times 10^{-5}$
WildJailbreak	vanilla-benign (210)	0.00%	6.19%	$b+c=13$, underpowered

Table 21: NotInject over-defense accuracy ($n = 339$ benign prompts), threshold 0.5. Higher is better. Wilson 95% intervals.

Model	Accuracy [95% CI]	FP	1 trig.	2 trig.	3 trig.
Hydra _{raw}	0.991 [0.974, 0.997]	3	0.991	0.991	0.991
Hydra	0.994 [0.979, 0.998]	2	1.000	1.000	0.982
ProtectAI v2	0.572 [0.519, 0.624]	145	0.779	0.478	0.460
InjecGuard*	0.873	—	—	—	—
PromptGuard*	0.009	—	—	—	—

*Reported by Li and Liu [2024], not re-measured here.

Table 22: Internal test set (19,030 examples), per head at threshold 0.5, on the original and relabelled label sets. Core macro- F_1 averages sql/pii/pi; toxic is advisory (Section 10).

Head	Original labels		Relabelled	
	Hydra _{raw}	Hydra	Hydra _{raw}	Hydra
SQL Injection	0.995	0.986	0.985	0.999
PII	0.970	0.910	0.910	0.992
Prompt Injection	0.764	0.847	0.618	0.860
Toxic (advisory)	0.790	0.457	0.488	0.927
Core macro-F_1	0.910	0.914	0.838	0.950

it subsumes two of our other benchmarks, our evaluations are not mutually independent; the Holm–Bonferroni family of Section 4 is a correction over related rather than independent tests.

Audit limitations. The cross-corpus comparison covers eight independent public corpora. WildGuard’s training mix is access-restricted and HackAPrompt and WildJailbreak were too large to complete within the run, so those three neither confirm nor refute the NotInject result; being the two largest available corpora, the latter two could still move it. The semantic tier (cosine) was not run in the cross-corpus pass. And the reference model’s corpus no longer exists, so model-versus-corpus overlap is measured against the current model’s corpus only. That is the correct direction for the threat tested — it removes rows the current model may have memorized — but “clean” is defined one-sidedly.

Residual overlap in the internal split. Our internal split was decontaminated from training by 13-gram and cosine filtering. Auditing it directly, the filter was incompletely applied: 10.6% of test rows share at least one word-13-gram with a training row and 137 (0.72%) are exact normalized duplicates, 5 with disagreeing labels; validation is comparable. The magnitude does not change the ordering in Table 22, and we report the internal set as a secondary result throughout.

Label-phrasing sensitivity. Both competitors’ scores move substantially with the label wording. Under a plain-language phrasing (“prompt injection or jailbreak attack”) Opir loses 0.253 AUPRC on `jackhhao` and begins scoring benign text highly (“What is the capital of France?” $\rightarrow$ 0.681). We report the favourable phrasing throughout, so the comparison is against each competitor at its best. Fixed heads with tuned thresholds do not have this degree of freedom, which is itself an argument for the output contract of Section 3.5.

Cross-lingual behaviour (Appendix B). The 250K-token multilingual embedding is 69% of the model’s parameters, and it earns that cost unevenly. On parallel machine-translated `xtram1` prompts the injection head’s AUPRC holds at 0.991–0.998 across six languages ($\Delta \leq 0.0085$ from the English anchor), but recall at the English-tuned threshold falls to 0.727–0.873 while precision stays at 1.000 — a threshold miscalibration, not a ranking failure, and one that per-language thresholds would largely recover. The toxic head genuinely degrades: on native Multilingual HateCheck, TPR at 1% FPR is 0.059 in Hindi and 0.161 in Arabic, i.e. unusable. SQLi transfer is unmeasured (the translator mangled the payload in 19–21 of 21 payload-bearing rows per language), locale-specific PII is untested, and no powered non-English *native* cell exists for either head, so the translated results stand without native corroboration.

References

- An, B. et al. Automatic pseudo-harmful prompt generation for evaluating false refusals in large language models. *arXiv preprint arXiv:2409.00598*, 2024.
- Barcha Correia, P. H. et al. A systematic literature review on LLM defenses against prompt injection and jailbreaking: Expanding NIST taxonomy. *arXiv preprint arXiv:2601.22240*, 2026.
- Chafik, S., Ezzini, S., and Berrada, I. Enhancing security in text-to-SQL systems: A novel dataset and agent-based framework. *Natural Language Processing*, 2025. Dataset licensed CC-BY-NC-ND.
- Cisco Security. Bypassing Meta’s LLaMA classifier: A simple jailbreak. Cisco Security Blog, 2025.
- Cui, J. et al. OR-Bench: An over-refusal benchmark for large language models. *ICML*, 2025. arXiv:2405.20947.
- Debenedetti, E. et al. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. *arXiv preprint arXiv:2406.13352*, 2024.
- Deepset. prompt-injections dataset. <https://huggingface.co/datasets/deepset/prompt-injections>, 2023.
- Dietterich, T. G. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923, 1998.

- Fomin, M. When benchmarks lie: Evaluating malicious prompt classifiers under true distribution shift. *arXiv preprint arXiv:2602.14161*, 2026. Unrefereed preprint.
- Geng, R. et al. PIArena: A platform for prompt injection evaluation. *arXiv preprint arXiv:2604.08499*, 2026.
- Greshake, K. et al. Not what you’ve signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. *arXiv preprint arXiv:2302.12173*, 2023.
- Han, S. et al. WildGuard: Open one-stop moderation tools for safety risks, jailbreaks, and refusals of LLMs. *NeurIPS Datasets and Benchmarks*, 2024.
- Hartvigsen, T. et al. ToxiGen: A large-scale machine-generated dataset for adversarial and implicit hate speech detection. *ACL*, 2022.
- He, P. et al. DeBERTaV3: Improving DeBERTa using ELECTRA-style pre-training with gradient-disentangled embedding sharing. *ICLR*, 2023.
- Inan, H. et al. Llama Guard: LLM-based input-output safeguard for human-AI conversations. *arXiv preprint arXiv:2312.06674*, 2023.
- Ivry, D. and Nahum, O. Sentinel: SOTA model to protect against prompt injections. *arXiv preprint arXiv:2506.05446*, 2025.
- Jackhhao. jailbreak-classification dataset. <https://huggingface.co/datasets/jackhhao/jailbreak-classification>, 2023.
- Jha, P. PII-Bench: A unified multi-source benchmark corpus for personally identifiable information detection. *arXiv preprint arXiv:2604.15776*, 2026.
- Kholkar, G. and Ahuja, R. CAPTURE: Context-aware prompt injection testing and robustness enhancement. *ACL LLMSec Workshop*, 2025. arXiv:2505.12368.
- Kumar, A. et al. Fine-tuning can distort pretrained features and underperform out-of-distribution. *ICLR*, 2022.
- Li, H. and Liu, X. InjecGuard: Benchmarking and mitigating over-defense in prompt injection guardrail models. *arXiv preprint arXiv:2410.22770*, 2024.
- Lin, Z. et al. ToxicChat: Unveiling hidden challenges of toxicity detection in real-world user-AI conversation. *EMNLP Findings*, 2023.
- Liu, Y. et al. Formalizing and benchmarking prompt injection attacks and defenses. *USENIX Security*, 2024.
- Meta. Llama-Prompt-Guard-2-86M. <https://huggingface.co/meta-llama/Llama-Prompt-Guard-2-86M>, 2025.
- Microsoft. Presidio: Data protection and de-identification SDK. <https://microsoft.github.io/presidio/>, 2023.
- Minko, B. et al. GLiNER Guard: Unified encoder family for production LLM safety and privacy. *arXiv preprint arXiv:2605.05277*, 2026. Unrefereed preprint.
- O’Neill, J. et al. Unified multi-task learning & model fusion for efficient language model guardrail. *arXiv preprint arXiv:2504.19333*, 2025.
- Padhi, I. et al. Granite Guardian. *arXiv preprint arXiv:2412.07724*, 2024.
- Panickssery, A. et al. LLM evaluators recognize and favor their own generations. *arXiv preprint arXiv:2404.13076*, 2024.
- Pedro, R. et al. From prompt injections to SQL injection attacks: How protected is your LLM-integrated web application? *ICSE*, 2025. arXiv:2308.01990.

- Perez, F. and Ribeiro, I. Ignore previous prompt: Attack techniques for language models. *arXiv preprint arXiv:2211.09527*, 2022.
- ProtectAI. `deberta-v3-base-prompt-injection-v2`. <https://huggingface.co/protectai/deberta-v3-base-prompt-injection-v2>, 2024.
- Ridnik, T. et al. Asymmetric loss for multi-label classification. *ICCV*, 2021.
- Röttger, P. et al. HateCheck: Functional tests for hate speech detection models. *ACL*, 2021.
- Saito, T. and Rehmsmeier, M. The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PLoS ONE*, 10(3):e0118432, 2015.
- Stepanov, I. and Smechov, A. Opir: Efficient multi-task safety classification for toxicity, jailbreaks, hate speech, and harmful content. *arXiv preprint arXiv:2605.29659*, 2026. Unrefereed preprint.
- Zeng, Y. et al. ShieldGemma: Generative AI content moderation based on Gemma. *arXiv preprint arXiv:2407.21772*, 2024.