



ON-PREMISE OPEN-WEIGHT LARGE LANGUAGE MODELS FOR INTERPRETABLE WEB VULNERABILITY DETECTION: A BENCHMARK AGAINST CLOUD MODELS AND RULE-BASED STATIC ANALYSIS

I. Shelestov^{1*}

¹Astana IT University, Astana, Kazakhstan, ORCID: <https://orcid.org/0009-0003-2398-7004>

*Corresponding author: Ilya Shelestov, shelestov2905@gmail.com

Abstract

Organisations handling regulated or proprietary source code are frequently prohibited from transmitting it to third-party inference providers, which excludes commercial cloud large language models from their security-analysis pipelines regardless of the accuracy those models achieve. Whether self-hosted open-weight models can substitute for them on ordinary developer hardware has not been established. This study benchmarks five quantised open-weight models, from 1.5 to 8 billion parameters and executed entirely on a consumer laptop graphics processor with eight gigabytes of video memory, on 120 annotated web-application code snippets spanning five weakness categories. Each was evaluated under zero-shot, few-shot and chain-of-thought prompting and compared against a rule-based static analyser and previously published cloud-model results on the identical corpus. The strongest local configuration reached an F1-score of 0.900, exceeding the previous-generation cloud flagship and trailing the current one by 4.7 points, while generating a remediation suggestion for every correct detection. The analysis further shows that F1 alone conceals degenerate behaviour: several configurations, including one cloud model, approach the score of a classifier labelling every input as vulnerable, so reporting specificity alongside F1 is proposed as a minimum requirement. Privacy-preserving on-premise vulnerability analysis is thus attainable without specialised infrastructure.

Keywords: *open-weight language models, web application security, vulnerability detection, on-premise deployment, model quantisation, explainability, static analysis, data sovereignty.*

1. Introduction

Web applications mediate access to banking, healthcare and electronic government services on which national digital infrastructure depends, and the weaknesses affecting them have proved remarkably stable: injection flaws, cross-site scripting, path traversal and cross-site request forgery still occupy the upper positions of industry weakness rankings despite decades of tooling investment. Static application security testing tools remain the standard automated defence, but



their integration into continuous-integration pipelines is impeded by persistent false-positive rates, by hand-authored rule sets that do not generalise beyond catalogued patterns, and by the absence of natural-language justification for their alerts [1], [2].

Large language models have, since 2022, become the dominant approach in the literature on automated vulnerability detection [3]. Their appeal is twofold: they capture semantic relationships that token-matching rules cannot express, and they emit human-readable explanations, addressing the interpretability deficit that limited earlier detectors [4]. Benchmarking studies report that general-purpose models under structured prompting outperform conventional analysers on intra-procedural weaknesses [5], [6], and an earlier evaluation by the present author and colleagues established F1-scores between 0.844 and 0.947 for the GPT-5 family on web-specific snippets, with the first systematic assessment of explanation quality for this task [7].

That prior work, however, shares a structural assumption with almost the entire literature: that the model is reached through a commercial application programming interface. This is untenable for a substantial class of users. Financial institutions, government contractors, defence suppliers and healthcare providers operate under regulatory regimes, confidentiality clauses or data-residency law forbidding transmission of proprietary source code to a third party. For them the accuracy of a cloud model is irrelevant, because it cannot be used at all; their realistic alternatives are a rule-based analyser inside their own perimeter, or a language model on hardware they control.

The second alternative has become plausible only recently, as quantisation compresses weights to four bits with modest quality loss and permissively licensed open-weight families now fit within the video memory of an ordinary workstation. Whether such models are accurate enough for security analysis remains unanswered: existing evaluations concentrate on C and C++ rather than web back-end languages, report detection accuracy without explanation quality, and use datacentre accelerators rather than hardware an organisation would deploy [8], [9], while work on compressed models such as Ibáñez-Lissen et al. [10] examines the compression mechanism rather than deployment.

This study therefore asks whether open-weight models executed entirely on commodity local hardware can perform interpretable web-vulnerability detection at a standard approaching that of commercial cloud models, and how both compare with a rule-based analyser on the same inputs. Four questions structure the investigation: what detection performance such models achieve; how their explanation quality compares with cloud models on the identical corpus; how both compare with rule-based analysis; and what throughput they sustain on a consumer graphics processor. The study contributes the first joint measurement of detection accuracy, explanation quality and deployment cost for self-hosted models on stated hardware, a three-way comparison on a single corpus, and the identification of a methodological hazard affecting the subfield.

2. Material and research methods

2.1 Evaluation corpus

The evaluation reuses, without modification, the corpus of 120 annotated code snippets built for that earlier cloud-model study [7], so that local and published cloud results can be compared without confound from differences in test data.

The snippets derive from the MITRE Common Weakness Enumeration [11], the YesWeHack vulnerable-code-snippets repository [12], and author-crafted examples following OWASP patterns [13]; safe counterparts apply the recommended remediation to a vulnerable example, following the paired-sample design of Wartschinski et al. [14]. The corpus spans nine languages — PHP (37), Python (34), Java (20), JavaScript (17), Perl (5), JSP (3), C# (2), SQL (1) and C (1) — each snippet carrying the ground-truth verdict, the weakness identifier and type, and the lines at which it manifests; Table 1 reports the composition.

Table 1. *Composition of the evaluation corpus by weakness category*

CWE	Weakness type	Vulnerable	Safe	Total
1	2	3	4	5
CWE-89	SQL injection	15	10	25
CWE-79	Cross-site scripting	17	9	26
CWE-78	OS command injection	15	8	23
CWE-22	Path traversal	14	10	24
CWE-352	Cross-site request forgery	13	9	22
Total		74	46	120

The class distribution — 74 vulnerable against 46 safe — is material to the interpretation of the results. A trivial classifier labelling every input as vulnerable attains a precision of 0.617, a recall of 1.000 and an F1-score of 0.763 on this distribution. This value is used throughout as a floor below which an F1-score carries no evidence of discriminative ability.

2.2 Systems under evaluation

Five models were selected to span three deliberate contrasts rather than to sample popularity: a code-pretrained model against a general-purpose model of comparable size; two members of one family sharing a training recipe but differing fivefold in parameter count; and a legacy code model retained because the governing research plan names it as a candidate base model. The roster comprises the Qwen2.5-Coder family [15], CodeLlama [16], Llama-3.1 [17] and Gemma-4. Selection was further constrained by the eight-gigabyte video-memory budget, which excluded current large code-specialised models requiring roughly eighteen gigabytes at four-bit precision. Table 2 reports the roster as deployed.

Table 2. *Open-weight models evaluated, as deployed*

Model	Parameters	Quantisation	On-disk size	Role in the design
1	2	3	4	5
Qwen2.5-Coder-1.5B	1.5 B	Q4_K_M	0.99 GB	Scale floor
Qwen2.5-Coder-7B	7.6 B	Q4_K_M	4.68 GB	Code-specialised
CodeLlama-7B-Instruct	7.0 B	Q4_0	3.83 GB	Legacy code model
Llama-3.1-8B	8.0 B	Q4_K_M	4.92 GB	General-purpose control
Gemma-4-E4B	7.5 B	Q4_0	6.15 GB	Current-generation general model



Two baselines complete the comparison. The rule-based baseline is the Semgrep static analyser [18], run in a container against the corpus materialised as source files, in three configurations: the security-audit ruleset, the OWASP Top Ten ruleset, and the union of nine registry rulesets, the last a best-effort rule-based result. The cloud baseline consists of the GPT-5.4, GPT-5.4-mini and GPT-5.1 figures previously reported on this corpus [7], reproduced for comparison and not re-measured.

2.3 Inference procedure

All local inference used the Ollama runtime [19] on a portable workstation with an NVIDIA GeForce RTX 5060 Laptop graphics processor providing 8151 MiB of video memory, 16 GB of system memory and Windows 11. No network egress occurred during inference; weights were retrieved once in advance.

Three prompting strategies were applied: zero-shot, few-shot with three worked examples, and chain-of-thought instructing an explicit source-to-sink trace before the verdict. The prompt texts are byte-identical to those of the cloud study, a precondition for the comparison, and each requests a JSON object containing the verdict, the weakness type and identifier, the vulnerable lines, an explanation and a suggested fix. Temperature was fixed at zero with a seed of 42, the context window at 8192 tokens and the generation limit at 1500 tokens; internal reasoning modes were disabled so that all models were exercised through the same interface. The design comprises 1800 inferences.

2.4 Response parsing

A naive extractor conflates two phenomena: a model emitting invalid JSON, and a harness failing to recover JSON that is present. Attributing the latter to the former would overstate the deficiency of local models, so a two-tier extractor was adopted. The first tier reproduces exactly the recovery ladder of the cloud study — fenced-block extraction, direct parsing, and a reverse brace scan — and defines the strict compliance rate reported against prior work. The second applies, only where the first fails, a string-aware balanced-brace scanner that does not terminate on braces or code fences inside string values, followed by repair of invalid escapes and unescaped quotation marks. Metrics use the best available tier, the strict rate being preserved as the comparable measurement; unrecoverable responses are excluded from the confusion matrix and reported explicitly.

2.5 Metrics

Detection is assessed by accuracy, precision, recall and F1-score with the confusion counts. Specificity, the proportion of safe snippets correctly cleared, is reported additionally; it was absent from the prior cloud study and is introduced here for reasons developed in Section 3.

Explanation quality is computed over true positives only, following the cloud study's definitions so that the figures are commensurable. Weakness-identifier accuracy is the proportion of true positives assigned the correct CWE identifier; weakness-type accuracy applies an alias mapping so that "XSS" and "cross-site scripting" are treated as identical. Line localisation is reported as exact set equality against the annotated lines and as partial overlap, and the fix rate as the

proportion of true positives accompanied by a remediation suggestion. Deployment cost is characterised by median latency and generation throughput, excluding cold-load queries.

3. The results and discussion

3.1 Detection performance

The experiment completed all 1800 inferences with no transport errors or empty responses. Table 3 reports detection performance for every model and strategy, ordered by F1-score.

Table 3. Detection performance of open-weight models on the evaluation corpus

Model	Strategy	Acc.	Prec.	Rec.	F1	TN/46	Spec.	Unrec.	Strict JSON
1	2	3	4	5	6	7	8	9	10
Gemma-4-E4B	CoT	0.867	0.837	0.973	0.900	32	0.696	0	0.967
Gemma-4-E4B	Zero-shot	0.842	0.887	0.851	0.869	38	0.826	0	0.975
Qwen2.5-Coder-7B	Zero-shot	0.792	0.782	0.919	0.845	27	0.587	0	0.917
Qwen2.5-Coder-7B	Few-shot	0.792	0.810	0.865	0.837	31	0.674	0	0.992
Gemma-4-E4B	Few-shot	0.783	0.816	0.838	0.827	32	0.696	0	0.992
CodeLlama-7B	Few-shot	0.703	0.710	0.978	0.822 [†]	1	0.053	56	0.525
Qwen2.5-Coder-7B	CoT	0.733	0.710	0.959	0.816	17	0.370	0	0.958
Llama-3.1-8B	Few-shot	0.708	0.697	0.932	0.798	16	0.348	0	1.000
Llama-3.1-8B	Zero-shot	0.667	0.655	0.973	0.783	8	0.174	0	0.867
Qwen2.5-Coder-1.5B	Zero-shot	0.642	0.635	0.986	0.772	4	0.087	0	0.900
Llama-3.1-8B	CoT	0.630	0.632	0.973	0.766	3	0.067	1	0.983
CodeLlama-7B	Zero-shot	0.617	0.617	1.000	0.763	0	0.000	0	0.950
Qwen2.5-Coder-1.5B	Few-shot	0.617	0.656	0.797	0.720	15	0.326	0	0.908
CodeLlama-7B	CoT	0.595	0.627	0.681	0.653 [†]	18	0.486	36	0.683
Qwen2.5-Coder-1.5B	CoT	0.542	0.627	0.635	0.631	18	0.391	0	0.975

[†] Computed on a reduced and non-random subset owing to unrecoverable responses; not comparable with the remaining rows.

Gemma-4-E4B under chain-of-thought prompting attains the strongest local result, with an F1-score of 0.900 and an accuracy of 0.867; its zero-shot configuration achieves a lower F1 of 0.869 but a markedly higher specificity of 0.826, a trade-off examined below. Qwen2.5-Coder-7B follows at 0.845 and Llama-3.1-8B at 0.798. CodeLlama-7B failed to produce recoverable output for 56 of 120 few-shot and 36 of 120 chain-of-thought responses; its apparent F1 of 0.822 is computed over the 64 parsed responses, and since parse failure is not independent of response



content that subset cannot be assumed representative, so both figures are marked non-comparable.

3.2 The F1-score conceals degenerate classifiers

On a corpus of 74 vulnerable and 46 safe snippets, a classifier returning "vulnerable" unconditionally achieves an F1-score of 0.763. Table 3 shows CodeLlama-7B under zero-shot prompting reproducing this value exactly, with 74 true positives, 46 false positives and no true negatives whatever: across 120 snippets it never once judged code safe. Its specificity is zero.

Three further configurations approach the same failure mode without reaching it outright: Llama-3.1-8B under chain-of-thought prompting cleared 3 of 46 safe snippets, Qwen2.5-Coder-1.5B under zero-shot prompting 4, and Llama-3.1-8B under zero-shot prompting 8. All record F1-scores between 0.766 and 0.783, within two points of the trivial baseline, by the same mechanism: near-indiscriminate positive prediction inflates recall while precision decays only as far as the class prior permits.

The hazard is not confined to small local models. Applied retrospectively to the prior cloud results, the measure reveals that GPT-5.1 under few-shot prompting attained a specificity of 0.435, clearing only 20 of 46 safe snippets, with an F1 of 0.844 that appears respectable; conversely GPT-5.4-mini under chain-of-thought prompting achieves the highest specificity of any language model in either study, 0.935. The distinction is invisible in the F1 column and obvious in the specificity column; Figure 1 plots both jointly.

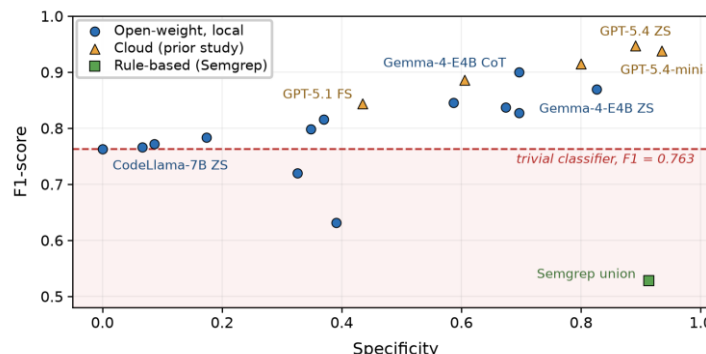


Figure 1. F1-score against specificity for all evaluated configurations. Points near the dashed line at low specificity score close to a classifier that labels every input as vulnerable.

The practical significance is considerable. A detector in a continuous-integration pipeline is judged by developers principally on the alerts that prove spurious, since these consume triage effort and erode confidence — precisely the complaint that motivated the search for alternatives to rule-based analysis. One with a specificity of 0.087 reproduces that complaint in a new form while presenting an F1-score suggesting competence. Specificity, together with a trivial-classifier baseline for the corpus in use, should therefore be reported alongside F1 as a matter of course. Read through this lens the ranking changes: only Gemma-4-E4B in all three configurations and Qwen2.5-Coder-7B in its zero-shot and few-shot configurations combine an F1 meaningfully

above 0.763 with a specificity indicating genuine discrimination, and these five constitute the usable results of the study.

3.3 Comparison with cloud models and rule-based analysis

Table 4 places the strongest configuration of each system alongside the previously reported cloud figures and the rule-based baseline.

Table 4. *Three-way comparison, best configuration per system*

System	Deployment	Strategy	Acc.	Prec.	Rec.	F1	Spec.
1	2	3	4	5	6	7	8
GPT-5.4	Cloud	Zero-shot	0.933	0.934	0.960	0.947	0.891
GPT-5.4-mini	Cloud	CoT	0.925	0.958	0.919	0.938	0.935
GPT-5.1	Cloud	CoT	0.891	0.886	0.946	0.915	0.800
Gemma-4-E4B	Local	CoT	0.867	0.837	0.973	0.900	0.696
Gemma-4-E4B	Local	Zero-shot	0.842	0.887	0.851	0.869	0.826
GPT-5.1	Cloud	Zero-shot	0.842	0.805	0.986	0.886	0.605
Qwen2.5-Coder-7B	Local	Zero-shot	0.792	0.782	0.919	0.845	0.587
Semgrep union	Local	9 rulesets	0.583	0.875	0.378	0.528	0.913
<i>Trivial classifier</i>	—	—	0.617	0.617	1.000	0.763	0.000

The principal finding is that a 6.15-gigabyte model file executing on a consumer laptop attains an F1-score 4.7 points below the current cloud flagship. Against the previous cloud generation the comparison is configuration-dependent: the local model exceeds GPT-5.1 under zero-shot prompting (0.900 against 0.886) and few-shot prompting (0.900 against 0.844), but does not reach its strongest configuration of 0.915. The capability that required a commercial inference contract at the time of the prior study is now approximately obtainable, at modest cost in accuracy, from hardware an organisation already owns. For a user prohibited from cloud inference the relevant comparison is not against GPT-5.4 but against the alternatives inside the perimeter, and against those the local model is superior by a wide margin.

The rule-based baseline fails in the exact inverse direction. Its configurations improve monotonically in recall as rules are added — the security-audit ruleset detects 9 of 74 vulnerable snippets (recall 0.122), the OWASP Top Ten ruleset 23 (0.311), and the union of nine rulesets 28 (0.378) — while precision remains between 0.875 and 0.900 and specificity between 0.913 and 0.978. Even at its most permissive it raises only four false alarms across the corpus, and when it fires its weakness classification is correct in 85.7 per cent of cases. The analyser is thus not weak but conservative: it reports little and is seldom wrong.

The explanation lies in the corpus. Registry rules for injection weaknesses are predominantly taint rules requiring a traceable path from an untrusted source to a dangerous sink, and decontextualised snippets frequently lack the enclosing request handler that would identify the source, so the rule cannot fire even where the weakness is plain to a reader. This is a genuine limitation of rule-based analysis on fragments rather than an artefact of configuration, and corresponds to a real scenario — reviewing a patch hunk in isolation — but these figures should not be read as



its performance on complete repositories. The two approaches thus fail in complementary directions: rule-based analysis under-reports and language models over-report. The natural synthesis, left to future work, is a pipeline trusting the analyser's high-precision findings directly while the language model adjudicates the remainder.

3.4 Explanation quality

Table 5 reports explanation quality over true positives, alongside the corresponding cloud figures.

Table 5. *Explanation quality on true positives, local and cloud*

System	Strategy	CWE acc.	Type acc.	Line exact	Line partial	Fix rate	n
1	2	3	4	5	6	7	8
GPT-5.4 (cloud)	CoT	0.957	0.986	0.300	0.771	1.000	70
GPT-5.4 (cloud)	Zero-shot	0.930	0.972	0.239	0.845	1.000	71
GPT-5.4-mini (cloud)	Few-shot	0.906	0.922	0.344	0.828	1.000	64
Gemma-4-E4B	Few-shot	0.855	0.855	0.290	0.726	1.000	62
Qwen2.5-Coder-7B	Zero-shot	0.853	0.853	0.147	0.412	1.000	68
Qwen2.5-Coder-7B	Few-shot	0.812	0.844	0.188	0.547	1.000	64
Gemma-4-E4B	Zero-shot	0.810	0.825	0.175	0.651	1.000	63
Gemma-4-E4B	CoT	0.750	0.750	0.208	0.625	1.000	72
Qwen2.5-Coder-1.5B	CoT	0.489	0.489	0.064	0.255	0.979	47

Weakness classification degrades gracefully with scale: the best local models reach 0.855, roughly ten points below the best cloud figure of 0.957, while the 1.5-billion-parameter model collapses to 0.489, indicating that below some threshold a model may detect that code is unsafe without reliably knowing why. Exact line localisation is poor everywhere and is not a local-model deficiency — the best local figure of 0.290 sits within the cloud range of 0.239 to 0.344 — so localisation should be presented as a hint rather than a determination; partial localisation separates the classes more clearly, with cloud models between 0.771 and 0.845 against a local best of 0.726.

The fix rate is the one measure on which local and cloud models are indistinguishable: every configuration except the smallest supplied a remediation suggestion for essentially every true positive. Since the practical value of a generative detector lies substantially in this capacity, its preservation under quantisation and local execution matters for the deployment argument.

A notable interaction appears between strategy and metric for Gemma-4-E4B: chain-of-thought prompting maximises detection F1 at 0.900 but yields its weakest classification accuracy at 0.750, whereas few-shot prompting reverses the ordering (F1 0.827, accuracy 0.855). Detection performance and explanation quality are therefore not jointly maximised by one strategy, and where explanations are surfaced to developers the few-shot configuration is preferable despite its lower F1.

3.5 Category-level performance

Figure 2 reports per-category F1-scores for each model's strongest configuration.

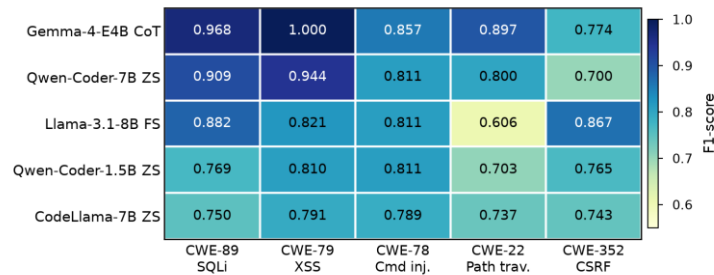


Figure 2. Detection F1-score by weakness category, best configuration per model.

SQL injection and cross-site scripting are detected most reliably, Gemma-4-E4B achieving perfect classification on the latter. Both present compact, conventional signatures — string concatenation into a query, unescaped interpolation into markup — well represented in code pretraining corpora. Path traversal and cross-site request forgery are the two weakest categories: for every model the lowest score falls to one of them, path traversal for three and cross-site request forgery for the other two. The latter, from 0.700 to 0.867, is limited for structural rather than capability reasons: establishing it requires knowing whether a state-changing request carries an unpredictable token and whether the framework enforces this by default, neither recoverable from an isolated snippet. Path traversal shows the widest spread of any category, 0.606 to 0.897, likewise reflecting dependence on surrounding validation. Together they mark the boundary at which snippet-level analysis ceases to be sufficient.

3.6 Deployment characteristics

Table 6 reports throughput and latency on the stated hardware, excluding cold-load queries.

Table 6. Inference throughput and latency on an 8 GB consumer graphics processor

Model	Output tok/s	Median latency, zero-shot	Few-shot	CoT	Total run time
1	2	3	4	5	6
Qwen2.5-Coder-1.5B	222	0.70 s	0.54 s	1.74 s	6.3 min
Gemma-4-E4B	87	2.82 s	1.75 s	9.25 s	29.0 min
Qwen2.5-Coder-7B	68	2.78 s	1.63 s	6.68 s	23.3 min
Llama-3.1-8B	65	4.28 s	1.70 s	7.96 s	29.0 min
CodeLlama-7B	25	18.17 s	9.19 s	20.58 s	100.9 min

The complete experiment occupied 3.14 hours and generated 570 800 output tokens, which at the published rate of the cloud flagship would cost approximately 8.56 United States dollars — modest at this corpus size, but scaling linearly with the volume scanned, whereas local inference cost is dominated by a fixed hardware investment already made. The more significant asymmetry is not financial: local inference emits no source code beyond the perimeter at any price.

Gemma-4-E4B, the most accurate local model, is also the second fastest at 87 output tokens per second and a median zero-shot latency of 2.82 seconds, so no accuracy-speed trade-off arises



in selecting it; at this rate a thousand-file scan completes in under an hour on one laptop, compatible with nightly pipeline execution though not with pre-commit hooks. CodeLlama-7B is dominated on every axis: the slowest by a factor of three and a half, the least compliant in structured output, and degenerate or non-comparable in every configuration. Since the governing research plan names it as a candidate base model, its underperformance is a useful negative result and a documented basis for selecting an alternative.

3.7 Structured output reliability

Strict JSON compliance among local models, excluding CodeLlama-7B, ranged from 0.867 to 1.000, against a cloud range of 0.950 to 1.000 measured with the identical first-tier extractor — close to indistinguishable, contradicting the expectation that motivated the two-tier design.

The second tier nevertheless proved necessary, recovering two to ten responses per model-strategy cell — correct analyses invalidated by three recurring defects: code fences inside the suggested-fix string, backslash sequences from PHP variable names that are not legal JSON escapes, and unescaped quotation marks in explanation text. Counting these as detection failures would have understated the affected models by a margin comparable to the differences the study seeks to measure. The recovered rate reached 1.000 for every model except CodeLlama-7B, whose 0.525 strict compliance is the sole genuine failure: it disregards the instruction entirely, returning discursive prose. Structured output is therefore reliable enough for pipeline integration provided the parser tolerates these defects, and compliance rates reported without describing the extractor are not comparable across studies.

3.8 Limitations

Three limitations bound these conclusions. The corpus consists of 120 short intra-procedural snippets, which suits controlled comparison but does not represent complete applications where inter-procedural data flow is decisive; the rule-based baseline is disadvantaged by this framing more than the language models are. Five weakness categories were examined, omitting broken access control and insecure deserialisation. Finally, all measurements derive from a single hardware configuration with eight gigabytes of video memory.

4. Conclusion

Five quantised open-weight models between 1.5 and 8 billion parameters were evaluated across three prompting strategies on 120 annotated snippets, and compared against a rule-based analyser and previously published cloud results on the identical corpus.

The answer is affirmative with qualifications. Gemma-4-E4B, a 6.15-gigabyte file on a consumer laptop graphics processor, reached an F1-score of 0.900, trailing the current cloud flagship by 4.7 points and within 1.5 points of the previous cloud generation's strongest configuration. It supplied a remediation suggestion for every correct detection and identified the weakness category correctly in up to 85.5 per cent of cases. For organisations barred from cloud inference, the choice is therefore no longer between accurate cloud analysis and no language-model analysis at all.

The methodological contribution is the demonstration that F1 is inadequate as a sole summary measure here: one model reproduced the trivial-classifier score of 0.763 exactly while never judging any code safe, and three further local configurations and one cloud configuration from prior work approached the same degenerate behaviour, invisibly in F1 and evidently in specificity. Specificity and a trivial-classifier baseline should be reported as standard on class-imbalanced corpora. Two further findings bear on practice: rule-based analysis and language models fail in opposite directions, recommending their combination rather than substitution; and detection accuracy and explanation quality are not maximised by the same prompting strategy.

For the wider dissertation, the results provide an evidence-based selection of Gemma-4-E4B as the base model for subsequent domain adaptation and a documented basis for rejecting CodeLlama-7B. The residual gap of 4.7 F1 points, with the specificity deficit of 0.696 against 0.891, defines the target that fine-tuning on a web-vulnerability corpus is intended to close. Further research should extend the evaluation to complete applications with inter-procedural data flow, broaden the weakness categories, validate explanation quality with practising analysts, and construct the hybrid pipeline the complementary failure modes suggest.

References

1. Chakraborty, S., Krishna, R., Ding, Y., & Ray, B. (2022). Deep learning based vulnerability detection: Are we there yet? *IEEE Transactions on Software Engineering*, 48(9), 3280–3296. <https://doi.org/10.1109/TSE.2021.3087402>
2. Harzevili, N. S., et al. (2024). A systematic literature review on automated software vulnerability detection using machine learning. *ACM Computing Surveys*, 57(3), 1–36. <https://doi.org/10.1145/3699711>
3. Zhou, X., Cao, S., Sun, X., & Lo, D. (2025). Large language model for vulnerability detection and repair: Literature review and the road ahead. *ACM Transactions on Software Engineering and Methodology*, 34(5), 1–31. <https://doi.org/10.1145/3708522>
4. Zhao, H., et al. (2024). Explainability for large language models: A survey. *ACM Transactions on Intelligent Systems and Technology*, 15(2), 1–38. <https://doi.org/10.1145/3639372>
5. Khare, A., et al. (2025). Understanding the effectiveness of large language models in detecting security vulnerabilities. In *2025 IEEE Conference on Software Testing, Verification and Validation (ICST)* (pp. 103–114). <https://doi.org/10.1109/ICST62969.2025.10988968>
6. Tamberg, K., & Bahşı, H. (2025). Harnessing large language models for software vulnerability detection: A comprehensive benchmarking study. *IEEE Access*, 13, 29698–29717. <https://doi.org/10.1109/ACCESS.2025.3541146>
7. Shelestov, I., Kanatova, A., Kali, M., Serik, D., Yessentay, N., & Zhetessova, G. (2026). Interpretable artificial intelligence models based on large language models for web application vulner-



- ability detection and security analysis. In *Proceedings of the International Conference on Computer Science, Digital Forensics and Artificial Intelligence (ICCSDFAI 2026)*. IEEE. <https://doi.org/10.1109/ICCSDFAI70505.2026.11647969>
8. Yin, X., Ni, C., & Wang, S. (2024). Multitask-based evaluation of open-source LLM on software vulnerability. *IEEE Transactions on Software Engineering*, 50(11), 3071–3087. <https://doi.org/10.1109/TSE.2024.3470333>
 9. Shestov, A., et al. (2025). Finetuning large language models for vulnerability detection. *IEEE Access*, 13, 38889–38900. <https://doi.org/10.1109/ACCESS.2025.3546700>
 10. Ibáñez-Lissen, L., González-Manzano, L., de Fuentes, J. M., & Anciaux, N. (2025). LPASS: Linear probes as stepping stones for vulnerability detection using compressed LLMs. *Journal of Information Security and Applications*, 93, 104125. <https://doi.org/10.1016/j.jisa.2025.104125>
 11. MITRE Corporation. (2026). *CWE — Common Weakness Enumeration*. <https://cwe.mitre.org/>
 12. YesWeHack. (2026). *Vulnerable-code-snippets* [GitHub repository]. <https://github.com/yeswehack/vulnerable-code-snippets>
 13. OWASP Foundation. (2026). *OWASP Foundation | Open source foundation for application security*. <https://owasp.org/>
 14. Wartschinski, L., Noller, Y., Vogel, T., Kehrer, T., & Grunske, L. (2022). VUDENC: Vulnerability detection with deep learning on a natural codebase for Python. *Information and Software Technology*, 144, 106809. <https://doi.org/10.1016/j.infsof.2021.106809>
 15. Hui, B., et al. (2024). Qwen2.5-Coder technical report. *arXiv preprint arXiv:2409.12186*. <https://doi.org/10.48550/arXiv.2409.12186>
 16. Rozière, B., et al. (2023). Code Llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*. <https://doi.org/10.48550/arXiv.2308.12950>
 17. Grattafiori, A., et al. (2024). The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*. <https://doi.org/10.48550/arXiv.2407.21783>
 18. Semgrep Inc. (2026). *Semgrep: Lightweight static analysis for many languages*. <https://semgrep.dev/>
 19. Ollama. (2026). *Ollama: Get up and running with large language models locally*. <https://ollama.com/>

Information about authors

Shelestov Ilya Andreevich –

Master's student, Astana IT University, Astana, Kazakhstan

e-mail: shelestov2905@gmail.com

ORCID: <https://orcid.org/0009-0003-2398-7004>