Activation Probes Surface Code-Security Signals that the Model's Output Misses

Ivan Wiryadi [1]

Abstract

AI coding agents now write a growing share of production code, and human security review does not scale at the rate code is generated. The agents in widest use are closed-weight, so a deploying team cannot read their internals. It can instead run an open-weight model as a reviewer over the agent's output. That reviewer's activations are readable. We ask whether reading those activations recovers a security signal that simply asking the same reviewer misses. We fit a single linear probe per model on a corpus of paired vulnerable-and-fixed Python functions, then test it without retraining on real disclosed vulnerabilities whose weakness type the probe never saw in training, across five open-weight reviewer models. On the vulnerabilities fixed by changing a single function, the probe scores the vulnerable function above its fix on 61–67% of cases for every model, beating the 50% chance line. It also beats the same model's prompted YES/NO win-rate read from its logits, under every prompt we try. Asking the model for a written verdict, even with chain-of-thought, returns the same answer on the vulnerable and fixed function most of the time and so cannot tell them apart. Model activations carry a code-security signal that prompting the same model misses.

1. Introduction

AI coding agents, such as *Claude Code*, *Cursor*, *Windsurf*, and *OpenAI Codex*, now author a growing share of production code (Favaro & Clark, 2026; Wang et al., 2025; Rooney, 2025). Empirical audits find that 24–29% of AI-assistant-generated Python and JavaScript snippets contain at least one security weakness, spanning tens of distinct weakness types (Fu et al., 2025). Human security review does not scale at the rate code is now generated, so some automated check in the loop is needed to triage what a reviewer looks at.

Current practice offers two such checks, and both are known to miss vulnerabilities. Static analyzers produce a high false-positive rate even on hand-crafted micro-benchmarks (Dubniczky et al., 2025), and prompting a coding LLM for a binary verdict is non-robust: trivial rewrites (renaming a variable, adding an unrelated library call) flip the answer in 17–26% of cases for GPT-4 and PaLM2 (Ullah et al., 2024). We confirm a related failure mode on paired vulnerable-and-fixed files: the model emits the *same* verdict on both most of the time, so its stated answer cannot tell them apart.

Linear probes on residual-stream activations encode safety-relevant signals that a model's own output can hide (McKenzie et al., 2025; Alain & Bengio, 2016; Orgad et al., 2025; Azaria & Mitchell, 2023; Marks & Tegmark, 2024). Similarly, LPASS applied linear probes on compressed-LLM activations to vulnerability classification on C/C++, training and testing on the same weakness types (Ibanez-Lissen et al., 2025). We extend this line of work along the axes that matter for a reviewer: we ask whether a linear probe recovers a security signal on Python that the model's own prompted output misses, and whether it transfers to weakness types held out of training and to real disclosed vulnerabilities. The signal is read straight off the model's hidden state. The longer-term goal is a control a team can self-host: an open-weight reviewer over a coding agent's output that flags changes worth re-reviewing, without access to the agent's own internals. Whether the signal can support that is what we begin to test here.

Across five open-weight code LLMs, the probe transfers to these unseen real-world vulnerabilities and outranks the same model's prompted output on them, a comparison we make precise in Section 2. Wiring the probe's verdict back into the coding agent is left to future work.

2. Method

Reviewer models. Five open-weight code LLMs spanning four architecture families: Qwen2.5-Coder 7B and

[1]Independent. Correspondence to: Ivan Wiryadi <inbox.ivn@gmail.com>.

Second Workshop on Technical AI Governance Research (TAIGR) @ ICML 2026, Seoul, South Korea. 2026. Copyright 2026 by the author(s).

14B (Hui et al., 2024), DeepSeek-Coder 33B (Guo et al., 2024), Devstral-Small-2505 (Mistral AI & All Hands AI, 2025), and Llama-3.1-8B (Grattafiori et al., 2024). Each is the instruct variant.

Data. We train on SVEN and evaluate on PATCHEVAL, both labelled by CWE. A CWE (Common Weakness Enumeration) is a *type* of software flaw, such as CWE-089 SQL injection; a CVE (Common Vulnerabilities and Exposures) is a single disclosed vulnerability, which carries one or more CWE labels. For training we use the four SVEN (He & Vechev, 2023) CWEs with substantial Python coverage (CWE-022, 078, 079, 089), which we call **SVEN-Python**; SVEN supplies each CWE as paired before-fix (vulnerable) and after-fix (secure) functions. For evaluation we use 234 Python CVEs from PATCHEVAL (Wei et al., 2025) whose bug type is *unseen*: no CWE in the CVE's full label set overlaps the four SVEN-training CWEs. (The remaining PatchEval CVEs, whose labels do overlap training, form a seen reference reported only in Appendix A.2.)

Probe fitting. For each model we extract the residual stream via `nnsight` (Fiotto-Kaufman et al., 2024), hooking the input of `layers[`L`].input_layernorm`. We sweep layers, token-pooling strategies (mean, max, last, sliding-window-max), and L_2 strength, selecting the configuration that maximises 5-fold `GroupKFold` AUC (Pedregosa et al., 2011). Both the train/eval split and the cross-validation folds are grouped by *project*: every repository lands entirely in train or entirely in eval. This keeps the probe from being scored on code it already saw in training, whether from the same project or the same vulnerability. Such duplication and leakage are known to inflate code-model evaluations (Allamanis, 2019; Ding et al., 2025). The selected layer and pooling vary by model; per-model choices are in Appendix A.1.

Out-of-distribution evaluation. We apply the SVEN-trained probes zero-shot to the unseen PATCHEVAL CVEs. For each CVE we score the vulnerable function ($\hat{p}_{\text{vul}}$) and its post-fix counterpart ($\hat{p}_{\text{fix}}$) and report the paired *win-rate*: the fraction of CVEs whose risk gap $\hat{p}_{\text{vul}} - \hat{p}_{\text{fix}}$ is positive, with exact ties dropped. We attach a Wilson 95% interval and an exact-binomial sign test against chance (0.5). The probe scores one function at a time, so a CVE fixed by changing a single function compares directly; these single-function CVEs are our headline slice and are unaffected by the pairing step below. When a patch touches several functions, we pair each vulnerable function with its fix by file path and definition name (dropping any vulnerable function with no matching fix) and average the matched-pair gaps into one risk gap per CVE, so the CVE stays the unit of analysis. We report the single-function slice and the full set separately.

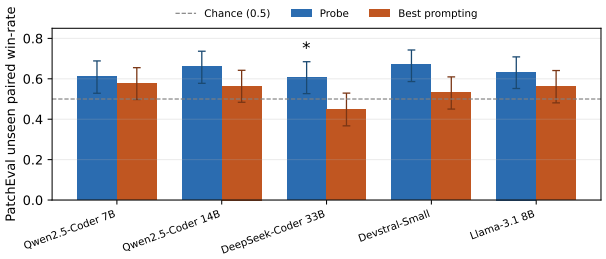

Figure 1. Paired win-rate on 147 unseen real-world vulnerabilities: probe versus the same model's best prompting, per reviewer model. A star marks a significant probe advantage over that model's *best* prompt-and-readout (paired McNemar, $p < 0.05$); Table 3 stars the advantage over each individual prompt mode.

Prompted comparator. Against the probe we run the same model's prompted output, varying the prompt three ways and reading the answer two ways. The prompt asks with no examples, with a few worked vulnerable/fixed examples from held-out weakness types (*few-shot*), or with a chain-of-thought trace before the verdict, generated with vLLM (Kwon et al., 2023). The *logit* reading takes the renormalised YES-versus-NO probability at the answer token (McKenzie et al., 2025), a continuous score comparable to the probe; as for the probe, a CVE is scored *correct* when this score is higher on the vulnerable function than on its fix. The *token* reading instead takes the literal "yes" or "no" string the model writes; here the two functions usually receive the same word, so the pair ties and cannot be ranked. We compare the probe against the strongest prompting per model with a paired McNemar test on the same CVEs. The strongest prompting is that model's highest-win-rate prompt-and-readout; picking the comparator's best configuration only makes the test harder for the probe. A tie counts as an incorrect prompting verdict.

3. Results

The probe scores vulnerable functions higher than their fixes on unseen bug types. We evaluate on 234 unseen real-world vulnerabilities from PATCHEVAL. Of these, 147 are fixed by changing a single function, the cleanest comparison. There the probe scores the vulnerable function above its fix on 61–67% of cases (Figure 1, Table 2). This fraction is the paired *win-rate*: how often the probe ranks the vulnerable function above its own fix. The Wilson 95% interval on it stays above the 50% chance line for every model, and the win-rate is above 0.5 by the sign test ($p < 0.05$ for every model). On the full set of 234 the win-rate is similar (62–69%). The effect is modest but holds across all five models and four architecture families. In-distribution, the same probe separates vulnerable from fixed functions at a fixed τ=0.5 threshold (AUC 0.77–0.85, accuracy 0.72–0.77; Table 1).

Table 1. In-distribution detection on SVEN-Python (n=152). Probe acc@0.5 is the probe's accuracy at a 0.5 threshold.

Model	Probe AUC	Probe acc@0.5	Prompted acc	Probe F1
Qwen2.5-Coder 7B	0.846	0.763	0.524	0.760
Qwen2.5-Coder 14B	0.821	0.737	0.518	0.733
DeepSeek-Coder 33B	0.844	0.770	0.488	0.755
Devstral-Small	0.771	0.737	0.500	0.747
Llama-3.1 8B	0.787	0.717	0.500	0.703

Table 2. Probe paired win-rate on the unseen-bug-types PATCHEVAL slice, single-function CVEs (n=147 pairs; no CWE overlaps the four SVEN-training CWEs). CI: Wilson 95%; p: sign test vs. 0.5. The seen-bug-types reference is in Appendix A.2.

Model	WR (%)	95% CI	p
Qwen2.5-Coder 7B	61.2	52.8–68.8	0.011
Qwen2.5-Coder 14B	66.2	57.8–73.7	2.4e-04
DeepSeek-Coder 33B	60.8	52.7–68.5	0.012
Devstral-Small	66.9	58.6–74.3	9.9e-05
Llama-3.1 8B	63.4	55.2–70.9	0.002

Table 3. Probe vs. prompted win-rate on unseen single-function vulnerabilities. *: probe significantly more accurate (paired McNemar, p<0.05).

	Probe	Prompting (%)		
Model	WR (%)	no-shot	few-shot	CoT
Qwen2.5-Coder 7B	**61.2**	43.8*	44.5*	57.8
Qwen2.5-Coder 14B	**66.2**	49.0	56.5	55.5
DeepSeek-Coder 33B	**60.8**	33.6*	35.4*	44.7*
Devstral-Small	**66.9**	48.6*	45.6*	53.1
Llama-3.1 8B	**63.4**	50.3	49.3*	56.2

The same model's prompted output does not. We prompt the same model three ways (no examples, a few examples, and chain-of-thought) and read its YES/NO answer two ways: from its logits or from the verdict it writes (Table 3). Reading the answer from the logits, the probe's win-rate is higher than the model's prompted output for every one of the five models, under each of the three prompts. By a paired McNemar test the probe significantly beats no-shot prompting on three of the five models and few-shot prompting on four; chain-of-thought read from the logits is significant on only one of the five, and the probe's win-rate is still higher for every model. We say nothing here about reading the written verdict, which ties on most pairs and so ranks too few to compare (next).

The logits separate the pair; the written verdict does not. The way the answer is read matters more than the prompt. Reading the YES-versus-NO probability off the logits gives a rankable score even when it is near chance, because the probability still differs between the vulnerable and the fixed function. Reading the verdict the model writes instead collapses: the model writes the *same* word for both functions on 72–97% of cases, so the pair ties and cannot be ranked.[1] The internal probability carries a signal the model's own stated answer throws away, the same gap the activation probe exploits more directly.

4. Discussion and Future Work

Future work. Wiring the probe into a coding agent's post-write hook, and evaluating it on the code those agents actually emit, is the natural next step. Our sweep covers mean, max, last, and sliding-window-max pooling with a logistic probe; attention-based probes and a probe-architecture comparison remain untried. Turning the paired signal into a calibrated detector with an absolute threshold is the main step toward deployment. Extension beyond Python is also open.

Limitations. Our out-of-distribution evaluation measures a paired comparison: whether the probe scores a vulnerable function above its own fix (win-rate 61–67%). On unseen weakness types this is a usable ranking signal. The probe leads every model on win-rate. Its advantage over the strongest prompting (chain-of-thought from the logits) is narrow and not always significant; the clear gap is over no-shot and few-shot prompting. We also worked with limited set data and expanding it is future work. Finally, we train only on the four SVEN CWEs with substantial Python coverage. The C/C++-dominant classes scored at chance in preliminary runs; whether that reflects a language, sample-size, or weakness-type effect is open.

LLM usage. Claude (Anthropic) was used extensively throughout the project for writing, editing, formatting, and coding, and for parts of the analysis. The authors set the design and framing decisions and reviewed outputs. Any remaining errors are the authors' own.

[1] Scored over the full slice, the written verdict is *correct* (YES on the vulnerable function and NO on its fix) on only 1–13% of pairs; on the rest the model writes the same verdict for both functions. Appendix A.3 (Table 6) breaks this down by prompt and readout.

References

Alain, G. and Bengio, Y. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*, 2016. URL https://arxiv.org/abs/1610.01644. ICLR 2017 Workshop.

Allamanis, M. The adverse effects of code duplication in machine learning models of code. In *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!)*, pp. 143–153. ACM, 2019. doi: 10.1145/3359591.3359735. URL https://arxiv.org/abs/1812.06469.

Azaria, A. and Mitchell, T. The internal state of an LLM knows when it's lying. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 967–976, 2023. URL https://aclanthology.org/2023.findings-emnlp.68/.

Ding, Y., Fu, Y., Ibrahim, O., Sitawarin, C., Chen, X., Alomair, B., Wagner, D., Ray, B., and Chen, Y. Vulnerability detection with code language models: How far are we? In *Proceedings of the IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pp. 1729–1741, 2025. doi: 10.1109/ICSE55347.2025.00038. URL https://arxiv.org/abs/2403.18624.

Dubniczky, R. A., Horvát, K. Z., Bisztray, T., Ferrag, M. A., Cordeiro, L. C., and Tihanyi, N. CASTLE: Benchmarking dataset for static code analyzers and LLMs towards CWE detection. arXiv preprint arXiv:2503.09433, 2025. URL https://arxiv.org/abs/2503.09433.

Favaro, M. and Clark, J. When AI builds itself. Anthropic Institute report, June 2026. URL https://www.anthropic.com/institute/recursive-self-improvement. Published 2026-06-04. As of May 2026, more than 80% of code merged into Anthropic's codebase was authored by Claude.

Fiotto-Kaufman, J., Loftus, A. R., Todd, E., Brinkmann, J., Juang, C., Pal, K., Rager, C., Mueller, A., Marks, S., Sharma, A. S., Lucchetti, F., Prakash, N., Brodley, C., Guha, A., Bell, J., Wallace, B. C., and Bau, D. NNsight and NDIF: Democratizing access to open-weight foundation model internals. *arXiv preprint arXiv:2407.14561*, 2024. URL https://arxiv.org/abs/2407.14561.

Fu, Y., Liang, P., Tahir, A., et al. Security weaknesses of Copilot-generated code in GitHub projects: An empirical study. *ACM Transactions on Software Engineering and Methodology*, 34, 2025. doi: 10.1145/3716848. URL https://arxiv.org/abs/2310.02059.

Grattafiori, A. et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. URL https://arxiv.org/abs/2407.21783.

Guo, D., Zhu, Q., Yang, D., Xie, Z., Dong, K., Zhang, W., Chen, G., Bi, X., Wu, Y., Li, Y. K., Luo, F., Xiong, Y., and Liang, W. DeepSeek-Coder: When the large language model meets programming – the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024. URL https://arxiv.org/abs/2401.14196.

He, J. and Vechev, M. Large language models for code: Security hardening and adversarial testing. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2023. URL https://arxiv.org/abs/2302.05319.

Hui, B., Yang, J., Cui, Z., Yang, J., Liu, D., Zhang, L., Liu, T., Zhang, J., Yu, B., Lu, K., et al. Qwen2.5-Coder technical report. *arXiv preprint arXiv:2409.12186*, 2024. URL https://arxiv.org/abs/2409.12186.

Ibanez-Lissen, L., Gonzalez-Manzano, L., de Fuentes, J. M., and Anciaux, N. LPASS: Linear probes as stepping stones for vulnerability detection using compressed LLMs. *arXiv preprint arXiv:2505.24451*, 2025. URL https://arxiv.org/abs/2505.24451.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, 2023. URL https://arxiv.org/abs/2309.06180.

Marks, S. and Tegmark, M. The geometry of truth: Emergent linear structure in large language model representations of true/false datasets. In *Conference on Language Modeling (COLM)*, 2024. URL https://arxiv.org/abs/2310.06824.

McKenzie, A., Pawar, U., Blandfort, P., Bankes, W., Krueger, D., Lubana, E. S., and Krasheninnikov, D. Detecting high-stakes interactions with activation probes. *arXiv preprint arXiv:2506.10805*, 2025. URL https://arxiv.org/abs/2506.10805. NeurIPS 2025.

Mistral AI and All Hands AI. Devstral-Small-2505, May 2025. URL https://huggingface.co/mistralai/Devstral-Small-2505. Model release, 2025-05-21.

Orgad, H., Toker, M., Gekhman, Z., Reichart, R., Szpektor, I., Kotek, H., and Belinkov, Y. LLMs know more than they show: On the intrinsic representation of LLM hallucinations. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2410.02707.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32 (NeurIPS)*, 2019. URL https://arxiv.org/abs/1912.01703.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, É. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. URL https://www.jmlr.org/papers/v12/pedregosa11a.html.

Rooney, K. Y Combinator startups are fastest growing, most profitable in fund history because of AI. CNBC, March 2025. URL https://www.cnbc.com/2025/03/15/y-combinator-startups-are-fastest-growing-in-fund-history-because-of-ai.html. Reports Y Combinator CEO Garry Tan: for ~25% of the Winter 2025 batch, 95% of lines of code are AI-generated.

Ullah, S., Han, M., Pujar, S., Pearce, H., Coskun, A., and Stringhini, G. LLMs cannot reliably identify and reason about security vulnerabilities (yet?): A comprehensive evaluation, framework, and benchmarks. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024. URL https://arxiv.org/abs/2312.12575.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., et al. SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, 17(3):261–272, 2020. doi: 10.1038/s41592-019-0686-2. URL https://doi.org/10.1038/s41592-019-0686-2.

Wang, B., Yu, W., Zhong, Y., Yu, H., Lian, K., Lu, C., Zheng, H., Zhang, D., and Li, H. AI code in the wild: Measuring security risks and ecosystem shifts of AI-generated code in modern software. *arXiv preprint arXiv:2512.18567*, 2025. URL https://arxiv.org/abs/2512.18567.

Wei, Z., Zeng, J., Wen, M., Yu, Z., Cheng, K., Zhu, Y., Guo, J., Zhou, S., Yin, L., Su, X., and Ma, Z. PATCHEVAL: A new benchmark for evaluating LLMs on patching real-world vulnerabilities. arXiv preprint arXiv:2511.11019, 2025. URL https://arxiv.org/abs/2511.11019.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45. Association for Computational Linguistics, 2020. URL https://aclanthology.org/2020.emnlp-demos.6/.

A. Appendix

A.1. Reviewer models and probe training

All five reviewer models are loaded with HuggingFace Transformers (Wolf et al., 2020) in float16, with no quantisation, on a single 80 GB H100; running every model at the same precision avoids a quantisation confound across the comparison. Activations are extracted with `nnsight` (Fiotto-Kaufman et al., 2024) at the input of `layers[`L`].input_layernorm`. Probes are trained in PyTorch (Paszke et al., 2019); layer-sweep selection and the held-out AUC use scikit-learn (Pedregosa et al., 2011); statistical tests use SciPy (Virtanen et al., 2020).

For each model we sweep the layer, the pooling strategy (mean, max, last, sliding-window-max), and the L_2 constant $C \in \{10^{-2}, 10^{-1}, 1\}$, choosing the configuration with the highest 5-fold `GroupKFold` AUC (Section 2). Table 4 reports the selected configuration, its cross-validation and held-out AUC on the 152-sample SVEN-Python eval split, and the same model's prompted YES/NO accuracy on that split for reference. At a 0.5 threshold the probe scores 0.72–0.77 accuracy, while the prompted YES/NO answer stays near chance (0.49–0.52), a gap of 21–28 points on the probe's own training distribution.

Table 4. Per-model probe setup and held-out SVEN-Python results. Pooling: max, or sliding-window-max with window 16 (swim). All models use $C{=}10^{-2}$. The prompted column is the same model's YES/NO accuracy on the SVEN-Python eval split.

Model	Layer	Pool	AUC (CV)	AUC (eval)	Prompted
Qwen2.5-7B	11	max	0.841	0.846	0.524
Qwen2.5-14B	21	swim	0.838	0.821	0.518
DeepSeek-33B	7	max	0.831	0.844	0.488
Devstral	21	max	0.846	0.771	0.500
Llama-3.1-8B	17	max	0.846	0.787	0.500

A.2. Seen-bug-types reference

The 169 PATCHEVAL CVEs whose CWE label set overlaps the four SVEN-training CWEs form a *seen* reference (the main text reports only the 234 unseen CVEs). This seen slice is the full seen set (single- and multi-function CVEs), whereas the main-text headline is the single-function unseen slice, so the two are not a strict like-for-like comparison. On the seen slice the probe's paired win-rate is higher (Table 5), as expected when the weakness type was in training; the unseen result in the main text is the generalisation claim.

Full unseen set. On the full set of 234 unseen CVEs (single- and multi-function), the probe's paired win-rate is 66.4% (Qwen2.5-Coder 7B), 69.4% (14B), 61.6% (DeepSeek-Coder 33B), 64.7% (Devstral-Small), and 66.7% (Llama-3.1 8B), all above 0.5 by the sign test ($p < 0.001$ for every model). This is the 62–69% range quoted in Section 3.

Table 5. Probe paired win-rate on the seen-bug-types PATCHEVAL slice ($n{=}169$ CVE pairs whose labels overlap the four SVEN-training CWEs). CI: Wilson 95%; p: sign test vs. 0.5.

Model	WR (%)	95% CI	p
Qwen2.5-Coder 7B	75.9	68.8–81.9	2.5e-11
Qwen2.5-Coder 14B	79.8	72.8–85.3	2.2e-14
DeepSeek-Coder 33B	64.5	57.0–71.3	2.0e-04
Devstral-Small	67.1	59.4–73.9	2.1e-05
Llama-3.1 8B	66.3	58.8–73.0	3.4e-05

A.3. Prompted comparator: *correct* vs *agree* on PatchEval

The prompt asks the model, as a security code reviewer, whether the function contains a vulnerability and to answer YES or NO; the few-shot prefix adds worked examples over weakness types held out of the eval set; the chain-of-thought variant asks for step-by-step reasoning ending in an explicit verdict. Out of the five models tested, four comply with the instruction, while DeepSeek-Coder 33B tends to refuse or fail to follow it more often (no-verdict rate 16% no-shot, 9% few-shot, 2% CoT), so its prompted win-rates are on a smaller compliant set.

For the prompted comparator we report the two paired quantities the main text turns on, on the unseen single-function PATCHEVAL slice. A pair is *correct* when the model ranks the vulnerable function above its fix, and the model *agrees* with itself when it assigns the *same* verdict to both functions (a tie). The two readouts behave oppositely. The continuous *logit* readout almost never ties (agree $\leq$1%), so its *correct* rate is measured over the full slice, and it sits at or below chance (34–58%). The binary *token* readout ties on most pairs (agree 72–97%), so its *correct* rate is computed over only the few non-tied remainder and is not comparable. The probe avoids this failure: it produces a continuous score on every pair, so there is always something to rank.

Table 6. Prompted comparator on the unseen single-function slice, as paired *correct* (ranks vulnerable above fix) and *agree* (same verdict on both) rates. Range across the five models. The *token correct* column is over the small non-tied remainder only.

Prompt $\times$ readout	Correct (%)	Agree / tie (%)
no-shot, logit	34–50	$\leq$1
few-shot, logit	35–57	$\leq$1
CoT, logit	45–58	0
no-shot, token	36–75	90–97
few-shot, token	10–85	86–93
CoT, token	40–65	72–86