

Where to Look: Energy-Based Fault Localization for Verus Vericoding

Guy Nachshon
Oz Labs
guy@ozlabs.ai

Apart $\times$ Atlas Computing · Secure Program Synthesis Hackathon, May 2026

Artifacts: github.com/ozlabsai/VericodingEBM **HF model:** [ozlabs/VericodingEBM](https://huggingface.co/ozlabs/VericodingEBM) **HF dataset:** [ozlabs/VericodingEBM](https://huggingface.co/datasets/ozlabs/VericodingEBM) **live demo**

Abstract

We study per-line fault localization for Verus, the Rust-based verified programming language. Our model is a Qwen2.5-Coder-1.5B LoRA specialist with sentinel tokens, per-line scalar energies, and a scalar attention-pool head trained on broken/fixed sibling pairs from the Microsoft Verus Training Data. On a marker-stripped corpus of 609 hand-authored failing Verus tests scraped from `verus-lang/verus`, the final scalar-head hybrid reaches AUROC = 0.78, top-1 = 0.56, and top-3 = 0.84. It beats six static baselines, but is matched or beaten by five zero-shot frontier LLMs on per-line ranking, pass/fail discrimination, and closed-loop CEGIS repair with the Verus toolchain in the loop. The contribution is therefore calibration rather than specialist superiority: we release the small-model baseline, line-labeled corpus, LLM and CEGIS records, and a strip-FAILS audit pipeline that exposed a `// FAILS/// FIXME` marker leak in an earlier checkpoint.

1 Introduction

Verified languages such as Verus [1] attach formal specifications to Rust-like code and discharge proof obligations to an SMT solver. Recent vericoding work shows that LLMs can draft Verus from natural-language specs and refine it from verifier feedback [2]. When a draft fails, a useful next signal is *where* to repair: a localizer could focus a CEGIS-style [3] repair loop on the top- k suspicious lines instead of resampling the whole implementation.

AI safety motivation. The safety case is scalable oversight for AI-written high-assurance software. As frontier models increasingly draft and repair code, formal verification can become an important guardrail for systems whose failures matter: access control, screening, lab automation, audit logging, and other high-consequence biosecurity infrastructure. But a verifier-in-the-loop workflow is useful only if humans and repair agents can audit where the proof or implementation is likely wrong. Line-level localization turns a failed proof into a ranked, inspectable set of places to scrutinize before accepting or repairing AI-generated code.

We cast this as discriminative energy-based modeling in the LeCun sense [4]. Given a spec and implementation, the model emits an unnormalized scalar energy for each scorable line and is trained from broken/fixed sibling structure in the Microsoft Verus Training Data. The final system is small enough to fine-tune with LoRA on one A100, yet produces a differentiable line-level suspicion field and a separately trained scalar head for whole-implementation ranking.

The central empirical story is mixed. The model is a strong static baseline, but frontier LLMs remain stronger, and our CEGIS experiment finds no repair-rate advantage. The audit story is equally important: an earlier

checkpoint looked much better until a strip-FAILS audit showed that Qwen’s pretrained comment-token priors were leaking through // FAILS/// FIXME markers.

Contributions.

1. **A discriminative energy-based per-line fault localizer for Verus.** The scalar-head hybrid (Qwen2.5-Coder-1.5B + LoRA + sentinel line energies) reaches AUROC = 0.78, top-1 = 0.56, and top-3 = 0.84 on the marker-stripped dev-test corpus ($n = 1,492$). Sections 3, 4.1 and 4.2.
2. **An architectural ablation isolating the load-bearing component.** Single-knob ablations attribute the +30pp AUROC gain over the adversarial-marker LSE design to the scalar attention-pool head; listwise loss and semi-hard mining help less, and logistic-vs-hinge is within noise (Section E, Table 13).
3. **A strip-FAILS audit pipeline.** We document a Qwen pretraining-prior leak that inflated an earlier checkpoint by 33pp top-3, then show that our fix overshot into marker-aversion while frontier LLMs are nearly marker-invariant (Sections 4.4 and 4.8).
4. **Comprehensive evaluation against external baselines.** The model beats every static baseline, but frontier LLMs match or beat it on ranking, discrimination, and CEGIS repair (Sections 4.3 and 4.5).

What is new in this hackathon. This submission’s new work is the Verus line-localization system and evaluation package: the dev-test scraper and line-labeled corpus, LoRA training runs for the LSE and scalar-head variants, the scalar-head hybrid architecture, the strip-FAILS audit, static/LLM/CEGIS baselines, plots, and report. We build on public Verus tooling, the Microsoft Verus Training Data, Qwen2.5-Coder, LoRA, EBMs, ListNet, and standard CEGIS ideas; those components are prior work, not claimed as new.

Scope. We evaluate per-line fault localization on hand-authored Verus failing tests and a closed-loop CEGIS repair-rate-at-1 experiment at $n = 100$. Transfer to Dafny [5], F*, or other verifiers is not addressed. All model numbers are from one training seed; bootstrap CIs capture corpus sampling variance, not seed variance.

2 Related Work

Process Reward Models (PRMs). PRMs score intermediate reasoning steps and aggregate them into an outcome score [6, 7, 8, 9]. Our setting is analogous, but the steps are Verus source lines and the supervision comes from broken/fixed sibling pairs rather than human labels or Monte Carlo rollouts. We also study the train/eval aggregator mismatch (“assessment drift”) highlighted by [10].

Energy-based models. We follow the discriminative EBM framework of LeCun et al. [4]: unnormalized scalar energy, contrastive training, and no generative sampling. Related recipes include JEM-style classifier-EBMs [11], compositional EBMs [12], and residual EBMs for text [13].

Fault localization. Spectrum-Based Fault Localization [14] (Ochiai, Tarantula, DStar) ranks lines by their differential presence in failing vs. passing runs. We adapt Ochiai to broken/fixed siblings as an oracle-style structural baseline; at deployment no PASS sibling exists, so it is not a deployable localizer.

Mutator validity. Just et al. [15] and Pradel and Sen [16] show that mutator-trained bug detectors can degrade sharply on real defects; Allamanis et al. [17] report similar gaps for self-supervised detectors. We therefore evaluate on hand-authored Verus failures rather than only mutator-generated held sets (Section 3.5).

3 Methods

3.1 Problem setting and notation

A spec-impl pair (s, c) consists of the spec block s (**requires**, **ensures**, **invariant**, **decreases** clauses plus the **fn** signature) and an implementation body c with scorable lines $c_1, \dots, c_N$. Each (s, c) has a binary outcome $y \in \{\text{pass}, \text{fail}\}$ from running Verus on the concatenation. For broken/fixed sibling pairs from **sft_safe** we additionally have $B \subset \{1, \dots, N\}$, the set of *buggy line indices* obtained by **diff1ib** comparison with the fixed sibling.

The model assigns to each line a scalar energy $E_\theta(c_i \mid s, c)$ (higher = more buggy), aggregates them via log-sum-exp at temperature T :

$$E_\theta(c \mid s) = T \cdot \log \left[\frac{1}{N} \sum_{i=1}^N \exp(E_\theta(c_i \mid s, c)/T) \right], \quad (1)$$

which interpolates between mean ($T \rightarrow \infty$) and max ($T \rightarrow 0$) and is mean-normalized so the aggregate does not scale with implementation length.

3.2 Architecture

The backbone is Qwen2.5-Coder-1.5B [18] with LoRA adapters [19] ($r=16$, $\alpha=32$) on all linear modules and a rank-8 LoRA on **embed_tokens**, $\sim 20\text{M}$ trainable parameters out of 1.56B total. A sentinel token (`<|fim_pad|>`, repurposed from Qwen’s FIM vocabulary) is inserted after each scorable impl source line; the per-line head MLP: $\mathbb{R}^{1536} \rightarrow \mathbb{R}$ (two-layer with GELU, dropout 0.1, hidden 256) reads each sentinel’s final-layer hidden state and emits an unbounded scalar energy. The sentinel’s causal attention sees the spec and all preceding impl lines.

Backbone choice. We chose Qwen2.5-Coder-1.5B for reproducibility and engineering fit, not because it was the newest or strongest code model available in 2026. Before training, we ran two tokenizer/license sweeps on a canonical Verus snippet using actual tokenizer probes. In the first sweep, Qwen2.5-Coder-1.5B/3B used 173 tokens and kept **requires**, **ensures**, **invariant**, and **decreases** as single tokens; StarCoder2-3B used 187 tokens, DeepSeek-Coder-V2-Lite 205, ModernBERT-base 193, UniXcoder-base 194, and CodeBERT-base 256. A broader second sweep found shorter candidate tokenizations from OpenCoder-1.5B-Base (167 tokens) and DeepSeek-Coder-6.7B-base (159 tokens), but the former required additional license review and the latter changed the compute and ablation budget. Qwen2.5-Coder-1.5B was therefore the stable primary: Apache-2.0, compact on Verus syntax, and FIM-pretrained in the same vocabulary we use for line sentinels. In particular, the technical report trains file- and repository-level fill-in-the-middle objectives with `<|fim_prefix|>`, `<|fim_middle|>`, `<|fim_suffix|>`, and `<|fim_pad|>` tokens; our sentinel is therefore not a newly added delimiter but a pretrained position type whose hidden state the backbone was already trained to make useful. The model is also small enough for LoRA sweeps and checkpoint release, and strong enough to test whether localization supervision teaches a capability beyond frozen-backbone surprisal. We evaluate newer frontier LLMs separately as external zero-shot baselines rather than as trainable specialist backbones.

3.3 Losses

We optimize $L = \lambda_{\text{spec}} L_{\text{spec}} + \lambda_{\text{line}} L_{\text{line}}$ with $\lambda_{\text{spec}} = 3$, $\lambda_{\text{line}} = 1$.

Within-spec InfoNCE. For each spec with mixed-label impls, the passing impl is the positive and failing siblings (plus an in-batch cross-spec FAIL pool) are negatives; the contrast is on whole-impl energies via Equation (1):

$$L_{\text{spec}} = -\log \frac{\exp(-E_\theta(c^+ \mid s))}{\sum_j \exp(-E_\theta(c_j \mid s))}. \quad (2)$$

Within-impl pairwise hinge. For each broken sibling with $|B| > 0$, we sample pairs (i, i') with $i \in B$ and $i' \notin B$ and penalize with margin $m = 1$:

$$L_{\text{line}} = \mathbb{E}_{(i, i')} \max(0, m + E_{\theta}(c_{i'}) - E_{\theta}(c_i)). \quad (3)$$

The two losses operate on disjoint subsets of the batch (`system_trajectory` triples for L_{spec} , `sft_safe` broken impls for L_{line}); the sampler enforces a minimum count of each per batch to avoid collapse.

Why not verifier-cited lines? We deliberately do not use Verus diagnostic line numbers as buggy-line labels. In verified-code failures, the diagnostic often cites the *consuming* assertion, postcondition, or invariant where the SMT obligation fails, not necessarily the source line that introduced the bad value or insufficient proof. In the trajectory files, FAIL rows also lack complete verifier logs, making diagnostic-derived line labels both noisy and incomplete. Line-level supervision therefore comes only from broken/fixed sibling diffs in `sft_safe/sft_part2`; diagnostic text is not used as a line-label source.

3.4 Data and split

A preliminary dataset audit found that most public “verified code” corpora are curated positives, theorem-proving traces, or prompt-only sets: the failing implementation attempts needed for contrastive localization have usually been filtered out. We therefore use the only public source we found with both mixed-label spec siblings and explicit broken/fixed repairs: four files from `microsoft/Verus_Training_Data`. The two trajectory files (`system_trajectory_843` and `algorithmic_trajectory_9040`) provide $281 + 6,402$ unique-spec impls at mixed pass/fail labels; the two SFT files (`sft_safe_25k` and `sft_part2_4557`) provide $\approx 14k$ broken/fixed sibling pairs with `difflib`-derived B . After parsing, filtering for $|B| \leq 3$ on `sft_*`, and tokenization to 4096-token context, the merged corpus contains 31,505 examples across 5,382 unique normalized-spec texts.

Spec-text-disjoint split. We hash spec text after whitespace normalization to derive a spec key, and split spec keys randomly into 29,525 train / 1,980 held-out examples. This is strictly stronger than splitting on the dataset’s `spec_id` field: we found that `spec_id`-based splits leak 79% of held-out spec content into train under different IDs, inflating AUROC to 0.98 by memorization. We assert post-split disjointness on normalized-text hashes.

3.5 Real-bug evaluation corpus

To test transfer beyond the mutator distribution, we constructed a Verus dev-test corpus from `verus-lang/verus` at HEAD. The repository’s test suite (`source/rust_verify_test/tests/*.rs`, 132 files) contains hand-authored test cases bracketed by `test_verify_one_file! { ... }` macros and paired with explicit outcomes (`=> Ok()` or `=> Err(...)`). Failing tests are annotated by Verus developers with `// FAILS` comments on the exact line that fails to verify—providing line-level ground truth that is *not* mutator-derived.

Parsing yields **1,492 records (609 FAIL with // FAILS labels)** across 132 test files. Mean impl length 19 lines, matching `sft_safe`’s 19.7 (no length-distribution confound). **We refer to this set throughout as the “Verus dev-test corpus.”** The bugs are hand-authored failing test cases written by Verus contributors, not production defects extracted from project bug reports; following [15] we avoid calling them “real bugs” to prevent confusion with the production-defect literature. They are nevertheless a substantially harder transfer target than mutator-induced failures (the model has never seen `verus-lang/verus` test sources during training), and to our knowledge this is the largest publicly available line-labeled Verus failure corpus — larger than the ~ 50 -bug Defects4J reference set used in earlier line-localization work [15], though the two are not strictly comparable (Defects4J indexes production Java defects across multiple repositories; ours is hand-authored failing tests from a single repository).

4 Results

4.1 Specialist performance on the marker-stripped Verus dev-test corpus (scalar-head hybrid)

The scalar-head hybrid (artifact run #10), our final architecture, achieves **AUROC** = 0.78, **top-1** = 0.56, **top-3** = 0.84 on $n = 1,492$ records (609 FAIL with labels) of the marker-stripped Verus dev-test corpus. Baselines are in Section 4.2; significance tests are in Table 11. An earlier checkpoint appeared to reach top-3 = 0.93; Section 4.8 explains why that number was marker-leaky.

4.2 Scalar-head hybrid vs. static and learned baselines (canonical comparison)

We compare against six execution-free baselines on the marker-stripped dev-test corpus ($n = 1,492$, $n_{\text{labeled}} = 609$):

RANDOM Uniform $[0, 1)$ per line; sanity floor.

LENGTH Rank longer lines higher.

VERUS KEYWORD Count occurrences of Verus DSL annotations: `assert`, `ensures`, `invariant`, `decreases`, `requires`, `forall`, `exists`, and `assume`.

MARKER KEYWORD Count occurrences of authorial bug markers: `// FAILS`, `panic!`, `unwrap`, `// TODO`, `assert!`, `todo!`, `unimplemented!`, `unsafe`, and `// FIXME`; this is the leak-ceiling baseline.

DIFF-FROM-SIBLING Rank target lines by $1 - \max_{\text{PASS sib}} \text{Jaccard}_{\text{bigram}}(\ell, \ell')$: a static adaptation of the NN-query baseline [20].

QWEN SURPRISAL Per-line average negative log-likelihood under the same Qwen2.5-Coder-1.5B backbone, frozen (no LoRA, no head). This is the code-naturalness floor [21, 22] and the most direct test of whether fine-tuning helps.

Figures 1 to 3 summarize the comparison; Table 11 gives paired McNemar and DeLong tests.

Headline finding. The scalar-head hybrid’s AUROC of 0.78 is +18pp over the frozen Qwen backbone (0.60; DeLong $p < 10^{-5}$) and +11pp over the strongest non-leak baseline (MARKER KEYWORD, 0.67; DeLong $p = 0.006$). On per-line top-3, the scalar-head hybrid (0.84) beats every non-leak baseline by $\geq +7\text{pp}$; the margin over frozen Qwen surprisal is +70pp.

Honest reporting: top-1 vs. Verus keywords. The one place where the scalar-head hybrid’s superiority is not statistically clean is top-1 against VERUS KEYWORD (0.560 vs. 0.544, McNemar $p = 0.53$). Both rank assertion-like lines high. The gap becomes decisive at top-3 ($p < 10^{-4}$), top-5 ($p < 10^{-2}$), and AUROC ($p < 10^{-15}$).

Training dynamics. The impl-pair (scalar-head) loss collapses to near-zero within the first 500 steps, while the per-line listwise+pairwise loss keeps improving through step 2,000. Whole-implementation discrimination is easier than localization in this setup.

4.3 Comparison to frontier zero-shot LLMs

We test five frontier LLMs via OpenRouter on the same marker-stripped Verus dev-test subset used for the scalar-head hybrid ($n = 220$: 200 FAIL +20 PASS, seed-fixed), with *two* prompts: (a) a **per-line ranking** prompt that asks for the top-5 most-suspicious line indices (Table 1); (b) a **discrimination** prompt that asks for {pass, fail} plus a calibrated $p(\text{fail})$ in $[0, 1]$ (Table 2, $n = 250$: 200 FAIL + 50 PASS, same FAIL subset). The LLMs see no `// FAILS` markers.

Headline: frontier zero-shot LLMs decisively outperform the scalar-head hybrid on both ranking and discrimination.

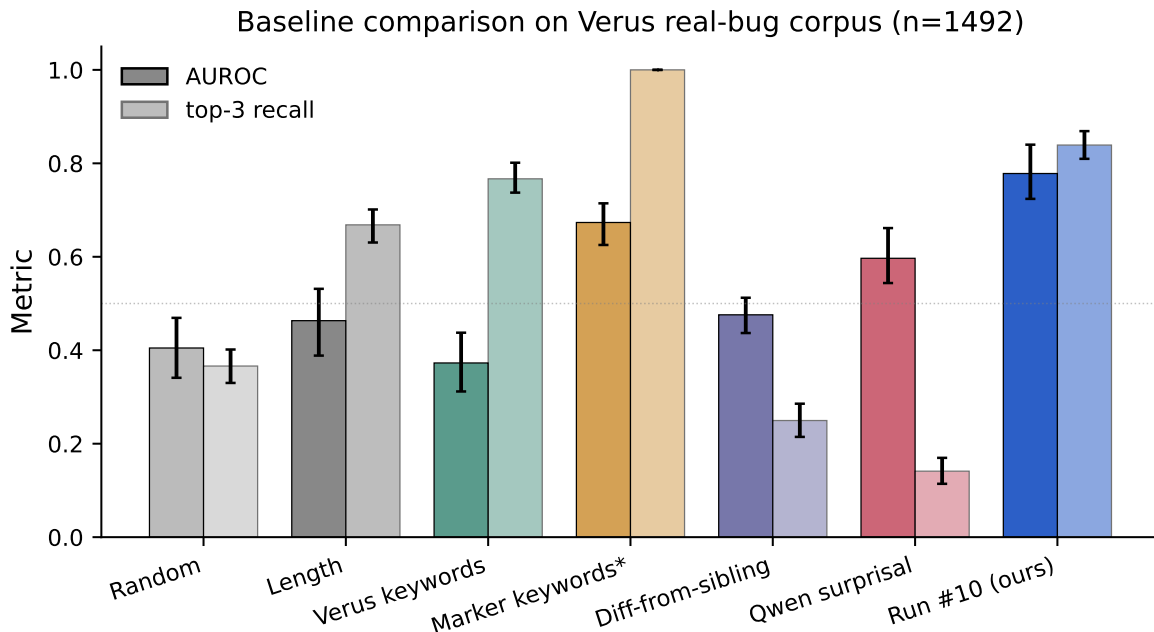


Figure 1: Scalar-head hybrid vs six static baselines on the marker-stripped Verus dev-test corpus ($n = 1,492$). Bars: AUROC (solid) and top-3 recall (faded); error bars are 95% bootstrap CIs over 300 resamples. The marker-keyword baseline (MARKER KEYWORDS*) attains top-3 = 1.00 by design (every record contains an authorial marker), but its impl-level AUROC of 0.67 is still substantially below the scalar-head hybrid’s 0.78 (DeLong $p = 0.006$). The scalar-head hybrid is the only method with both AUROC > 0.7 and top-3 > 0.8.

Reading the result honestly. On per-line top- k , Claude, Qwen, and Gemini reach top-3 ≥ 0.97 vs. the scalar-head hybrid’s 0.82, and top-1 ≥ 0.86 vs. 0.56. On whole-impl AUROC, the gap is also large, though operationally less important because the verifier itself supplies pass/fail with AUROC = 1.0. We use AUROC mainly as a sanity check that the scalar head learned a coherent implementation-level signal.

What the table does *not* establish. This comparison does not rule out specialist value under distribution shift, API-cost constraints, self-judging-loop concerns, or future Verus syntax. It does establish the negative calibration result on this corpus: single-call frontier LLM localization is stronger.

Reproducibility note. Ranking-prompt model calls were made via OpenRouter on 2026-05-23 12:00–14:00 UTC; discrimination-prompt calls on 2026-05-23 22:55–23:25 UTC with explicit OpenRouter slugs and temperature = 0. Numbers may shift if providers update snapshots; rerunnable scripts are `scripts/baseline_llm_zeroshot.py` (ranking) and `scripts/baseline_llm_discrimination.py` (discrimination).

4.4 Marker-leak audit: invariance, reliance, and aversion

The strip-FAILS audit asks how a localizer’s top- k ranking changes when the `// FAILS` substring is removed at evaluation. We run it on the same $n = 220$ subset for the three strongest LLMs, on the marker-leaky LSE checkpoint (run #7, full corpus), and on the post-fix scalar-head hybrid.

Three regimes; the specialist never reaches invariance. Frontier LLMs are essentially marker-invariant ($|\Delta| \leq 3.5\text{pp}$ top-1). The marker-leaky LSE checkpoint is marker-reliant: stripping markers drops top-1 by 46pp. The scalar-head hybrid is marker-averse: keeping markers drops top-1 by 50pp below its stripped performance, to 0.055, below the uniform-random floor. The adversarial marker training broke reliance but overshot into anti-correlation.

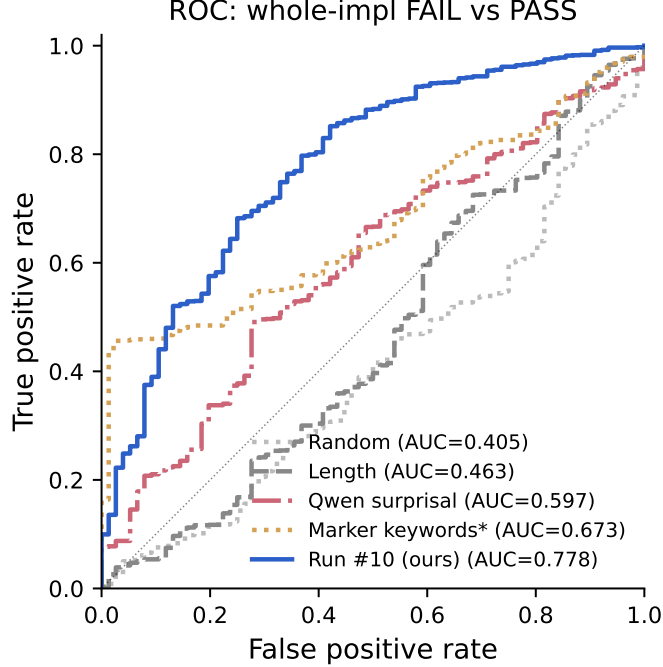


Figure 2: ROC curves for whole-impl FAIL vs. PASS discrimination (scalar-head hybrid, stripped, vs. four representative baselines). The scalar-head hybrid dominates uniformly at every false-positive rate.

What this means. The headline stripped numbers in Table 12 are the right deployment metric for marker-free inputs, but a leak-immune localizer should sit near $\Delta = 0$. Frontier LLMs do; both trained specialists miss in opposite directions.

4.5 Closed-loop CEGIS: does the specialist help *repair*?

Static localization is only useful if it improves repair. We therefore run a closed-loop CEGIS-style comparison, with the Verus toolchain adjudicating success, analogous in spirit to fault-aware code editing systems such as Self-Edit [28].

Setup. On $n = 100$ tractable single-buggy-line FAIL implementations (≤ 20 scorable lines, ≤ 25 source lines), we run three Claude Opus 4.7 repair arms and adjudicate each with **verus** v0.2026.05.17:

Arm A (specialist-guided): Feed (spec, impl, the scalar-head hybrid’s top-3 lines) to Claude Opus 4.7 with a “rewrite the flagged lines to make Verus accept it” prompt. Wrap the response, invoke Verus.

Arm B (LLM-only resample): Feed (spec, impl) to Claude with a “rewrite the implementation to make Verus accept it” prompt — no localization hint. Wrap, invoke Verus.

Arm C (LLM-self-judged): Two-pass Claude. First call: Claude ranks its own top-5 most-suspicious lines (same prompt as the ranking baseline in Table 1). Second call: identical to Arm A but with Claude’s own top-3 flagged instead of the specialist’s. Wrap, invoke Verus.

Arm A tests specialist guidance, Arm B tests LLM-only resampling, and Arm C tests whether Claude can supply its own localization hint.

Honest reading. All three arms cluster at 25–30% repair-rate, within each other’s 95% bootstrap confidence intervals (roughly $\pm 9\text{pp}$). **No arm is statistically separable at $\alpha = 0.05$.** The point estimates are

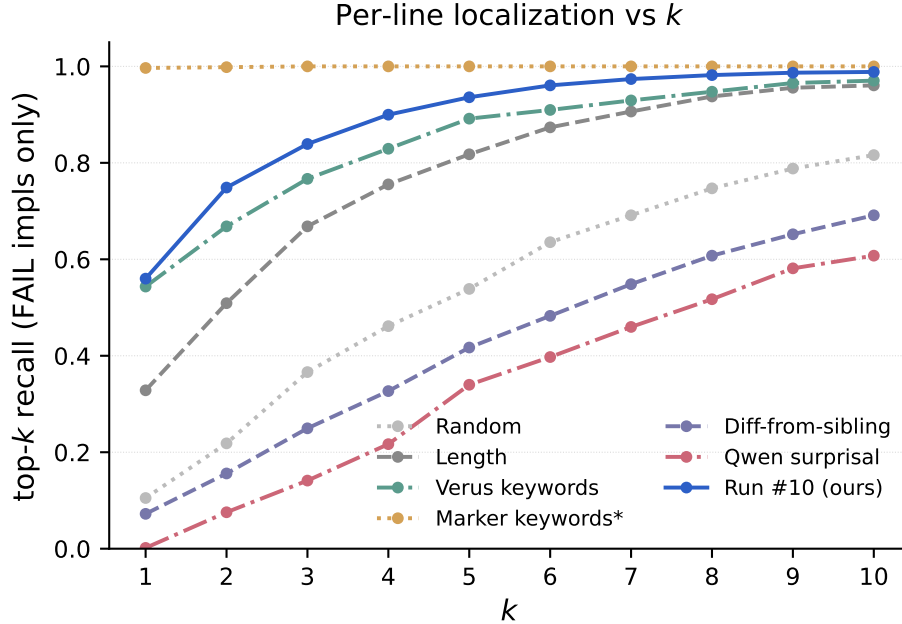


Figure 3: Per-line localization recall as a function of k . The scalar-head hybrid reaches top-3 = 0.84 where the strongest non-leak baseline (Verus keywords) needs $k = 5$ to match. The marker-keyword leak ceiling saturates at 1.00 for all $k \geq 1$ (and motivates the strip-FAILS audit).

directionally against the specialist, but the experiment is underpowered; the 0 specialist-only wins in A-vs-C disagreements ($p = 0.06$) is a follow-up hypothesis, not a claim.

Structural finding. The arms are not interchangeable. All three solve 21 impls, all three fail on 65, and at least one arm solves 35. This leaves possible headroom for routing or ensembling, but the 35% union assumes an oracle router.

Methods footnote: Arm B prompt fix. The initial Arm B prompt omitted the failing impl and caused 38/100 empty code blocks. We reran Arm B with the failing impl as anchor, matching Arms A and C; empty-block returns fell to 3, and repair-rate moved from 0.27 to 0.30. Both prompt versions are released in [artifacts/cegis/](#).

What this means. On this corpus at this scale, specialist-guided repair does not produce a detectable advantage over LLM-only or LLM-self-judged repair. Per-line ranking, discrimination, and closed-loop repair all point in the same direction.

4.6 Whole-impl AUROC ceiling: an aggregator-invariance observation

Whole-impl AUROC is sensitive to how line energies are aggregated. To test whether the LSE head was simply the wrong reducer, we rescored the same per-line outputs with standard permutation-invariant aggregators. All remain near chance: mean, max, min, sum, last, and softplus-sum land in $[0.47, 0.53]$ (Table 5). This supports the view that bag-of-lines aggregation is the bottleneck for the LSE checkpoint.

For the scalar-head hybrid we report only the trained scalar attention-pool head. Choosing a different reducer at evaluation time would be assessment drift [10]; the sweep is a diagnostic, not a headline metric.

Table 1: **Per-line ranking.** Frontier LLMs zero-shot vs. the scalar-head hybrid on the same $n = 220$ subset of the marker-stripped Verus dev-test corpus, ranking-prompt. **Bold** = best in column. “Failures” counts records where the model returned non-parseable output.

Method	top-1	top-3	top-5	failures
Random baseline	0.11	0.37	0.54	—
Scalar-head hybrid (ours, 1.5B)	0.56	0.82	0.93	—
Claude Opus 4.7 [23]	0.86	0.97	0.99	0 / 220
GPT-5.5 [24]	0.85	0.92	0.93	24 / 220
Gemini 3.5 Flash [25]	0.89	0.98	1.00	0 / 220
Qwen 3.7 Max [26]	0.91	0.99	0.99	0 / 220
DeepSeek V4 Pro [27]	0.45	0.61	0.67	90 / 220 (41%) [†]

[†] DeepSeek V4 Pro’s 41% parse-failure rate is a budgeting artifact (its hidden chain-of-thought consumed the 1500-token output budget); we report the number for completeness but note it is not a capability finding.

Table 2: **Discrimination AUROC.** The same five LLMs prompted with a $\{\text{pass}, \text{fail}, p(\text{fail}) \in [0, 1]\}$ prompt on $n = 250$ records (200 FAIL + 50 PASS, marker-stripped). AUROC is computed over $p(\text{fail})$ against the true status; verdict accuracy is exact-match against status. **Bold** = best in column. Frontier zero-shot LLMs decisively outperform the scalar-head hybrid on this metric too.

Method	AUROC	verdict acc.	parse failures
Random baseline	0.50	—	—
Scalar-head hybrid (ours, 1.5B)	0.78	—	—
Claude Opus 4.7 [23]	0.977	0.968	0 / 250
Qwen 3.7 Max [26]	0.946	0.920	0 / 250
Gemini 3.5 Flash [25]	0.983	0.956	0 / 250
GPT-5.5 [24]	0.984	0.981	38 / 250
DeepSeek V4 Pro [27]	0.914	0.914	64 / 250

4.7 Statistical significance and subgroup analysis

Significance vs. the length baseline. Table 11 reports paired McNemar tests for top- k and paired DeLong tests for AUROC. The audited scalar-head hybrid is decisively better than length and frozen-Qwen baselines on top-3 and AUROC, while top-1 against Verus-keyword is not significant. The older pre-audit LSE-vs-length contingency is retained in the released artifacts for historical comparability.

Bug-type breakdown. We bucket the Verus dev-test FAIL set by the syntactic context of the `// FAILS-` annotated line (regex match on the line text):

The model is strong on **assert** and non-keyword lines, but **ensures**-clause failures drop to top-3 = 0.59 on $n = 22$.

Why ensures-clauses are hard, mechanistically. The likely reason is architectural. A sentinel at line ℓ reads a local causal hidden state, which works for assertions and arithmetic whose signal is co-located with the line. Postcondition violations can surface at an **ensures** line even when the causal explanation is distributed upstream. A global-attention head alongside the local sentinel head is the most direct follow-up.

Robustness across $|B|$ and impl length. We stratify the same Verus dev-test FAIL set two further ways:

Top-3 is robust across buggy-line count (no drop for hard $|B| = 1$ vs. easier $|B| = 2$). It degrades mildly with length, from 0.985 on short impls to 0.897 on long ones.

Table 3: Marker-leak audit, three regimes. “With markers” keeps the `// FAILS` comment in the impl shown to the localizer; “stripped” removes it; Δ = with-minus-stripped, so *positive* Δ = marker-reliant (signal goes *down* when marker is removed), *negative* Δ = marker-averse (signal goes *up* when marker is removed). LLM and scalar-head hybrid numbers on the $n = 220$ subset (same as Table 1); marker-leaky LSE numbers on the full $n = 1,492$ corpus where the leak signal is largest (reproduced from Table 8).

Model	Setting	top-1	top-3	top-5
Frontier LLMs (essentially marker-invariant)				
Claude Opus 4.7	with markers	0.870	0.955	0.970
	stripped	0.860	0.970	0.985
	Δ	+0.010	−0.015	−0.015
Qwen 3.7 Max	with markers	0.920	1.000	1.000
	stripped	0.905	0.985	0.985
	Δ	+0.015	+0.015	+0.015
Gemini 3.5 Flash	with markers	0.925	1.000	1.000
	stripped	0.890	0.980	1.000
	Δ	+0.035	+0.020	+0.000
Marker-leaky LSE (artifact run #7; marker-reliant)				
Marker-leaky LSE (run #7)	with markers	0.73	0.93	0.96
	stripped	0.27	0.60	0.76
	Δ	+0.46	+0.33	+0.20
Scalar-head hybrid (artifact run #10; marker-averse)				
Scalar-head hybrid	with markers	0.055	0.265	0.535
	stripped	0.560	0.815	0.930
	Δ	−0.505	−0.550	−0.395

4.8 Strip-FAILS audit: where the pre-audit 0.93 number came from

The marker-leaky LSE checkpoint (run #7) appeared to reach top-3 = 0.93 on hand-authored Verus failures, suspiciously high for a model trained on mutator data [15, 16, 17]. We audited this by re-scoring the same corpus after stripping `// FAILS` comments at evaluation time.

The smoking gun: stripping markers collapses transfer. With markers retained, top-3 is 0.93; stripped, it falls to 0.60, below the length baseline. The `// FAILS` substring carried a large share of the apparent transfer.

Mechanism: a Qwen pretraining-prior leak through the FAILS/FIXME embeddings. The training corpus contains no `// FAILS` markers, so the LoRA finetune had no reason to override Qwen’s pretrained association between tokens such as FAILS/FIXME and suspicious code. Embedding-surgery confirms the mechanism (Section E); the training-time mitigation is adversarial marker injection.

Alternative leak hypotheses. We ran targeted falsification tests for four alternatives.

No whole-text contamination. Hash-based exact-match between the Verus dev-test corpus and the full 31,505-example training set: **0/1492** spec-text matches and **0/1492** impl-text matches (whitespace-normalized).

Line-level leakage exists but does not drive the result. Some lines do recur: 39% of scorable lines and 20.8% of comment-stripped buggy lines appear verbatim somewhere in training. But the conditional test below shows overlap is not higher on hits.

This is the opposite of the memorization pattern.

Table 4: Closed-loop CEGIS repair-rate-at-1 on $n = 100$ tractable FAIL impls from the Verus dev-test corpus, adjudicated by the actual Verus toolchain. Pairwise McNemar exact two-sided p -values on discordant pairs; all three arms cluster at 25–30% with no statistically separable differences at $\alpha = 0.05$.

Arm	Repair-rate-at-1
A. Specialist top-3 $\rightarrow$ Claude rewrite	0.25
B. LLM-only (Claude resample, no hint)	0.30
C. LLM-self-judged (Claude localize $\rightarrow$ rewrite)	0.30
<i>Pairwise discordance (only-A / only-other; McNemar p):</i>	
A vs. B	4 / 9, $p = 0.27$
A vs. C	0 / 5, $p = 0.06$
C vs. B	5 / 5, $p = 1.00$
Union (any arm solved)	0.35
Intersection (all arms solved)	0.21

Table 5: Whole-impl AUROC on Verus dev-test corpus under different eval-time aggregators of the *same* per-line energies. All reducers cluster around chance, supporting the information-theoretic ceiling argument.

Aggregator (eval-time, same per-line energies)	AUROC
mean	0.471
max	0.484
min	0.490
last	0.517
sum	0.530
softplus-sum	0.475
LSE (train-time matched)	0.487

Verus-keyword baseline. A DSL-aware baseline ranks by Verus keyword count. It is stronger than length but does not explain the model’s pre-audit top-3 result:

Position-distribution shift survives. `sft_safe` buggy lines cluster early (median 0.31); dev-test bugs cluster late (median 0.75). The model’s predictions track the dev-test distribution (median 0.67), not the training prior, and energy-position Spearman averages +0.03 (Figure 4).

Summary of the audit. The audit identifies marker leakage as the driver and rules out exact contamination, simple line memorization, keyword-only behavior, and a position prior. Seed variance remains a limitation (Section 5). The script `scripts/strip_fails_reeval.py` is released for re-auditing future models with the same record schema.

Table 6: Top-3 by lexical category of the `// FAILS`-annotated line on the Verus dev-test corpus. Categories with $n \leq 1$ omitted. **other** denotes lines that do not lexically contain any of the listed Verus keywords (typical content: variable assignments, arithmetic expressions, method calls). **ensures** is the model’s weakest subgroup—a paper-worthy limitation discussed in Section 5.

Category at <code>// FAILS</code> line	n	top-3
assert	369	0.943
other	215	0.949
ensures	22	0.591

Table 7: Top-3 stratified by buggy-line count ($|B|$) and impl line-count tertile, on the Verus dev-test corpus ($n = 609$).

$ B $ stratum	n	top-3	Length stratum	n	top-3
$ B =1$	455	0.921	short (≤ 8 lines)	203	0.985
$ B =2$	73	0.973	medium (8–16)	203	0.911
$ B \geq 3$	81	0.951	long (16–139)	203	0.897

Table 8: Strip-FAILS audit on the Verus dev-test corpus ($n=1,492$, run #7 step-500 checkpoint). The marker provides substantial discriminative signal: top-3 falls 33pp when markers are stripped, dropping below the length baseline. We confirm the audit generalizes by also stripping markers from baselines (which mostly do not use them) and from the marker-keyword regex (which uses them by construction — bold row).

	With // FAILS	Stripped
top-1	0.73	0.27 (−46pp)
top-3 (marker-leaky LSE, run #7)	0.93	0.60 (−33pp)
top-5	0.96	0.76 (−20pp)
AUROC (LSE)	0.49	0.44
Length baseline top-3	0.73	0.46
Keyword baseline top-3	0.80	0.55

Table 9: Conditional train-overlap rate, split by whether the model’s top-1 prediction was correct on dev-test failures. If the model were winning by memorization, HITs would show HIGHER train-overlap than MISSES. The opposite holds.

	top-1 HIT ($n=445$)	top-1 MISS ($n=164$)
pred-top1 in training corpus	20.9%	27.4%
ground-truth buggy line in training corpus	24.0%	23.2%

Table 10: Verus-keyword baseline on both corpora, vs. length baseline and our model. Keyword baseline beats length top-3 on both corpora, but the model retains a +17pp (held) / +13pp (real) gap.

	Held top-3	Dev-test top-3
Uniform random	0.40	0.38
Length	0.74	0.73
Verus-keyword (this section)	0.73	0.80
Ours (LSE)	0.90	0.93
gap vs. keyword	+17pp	+13pp

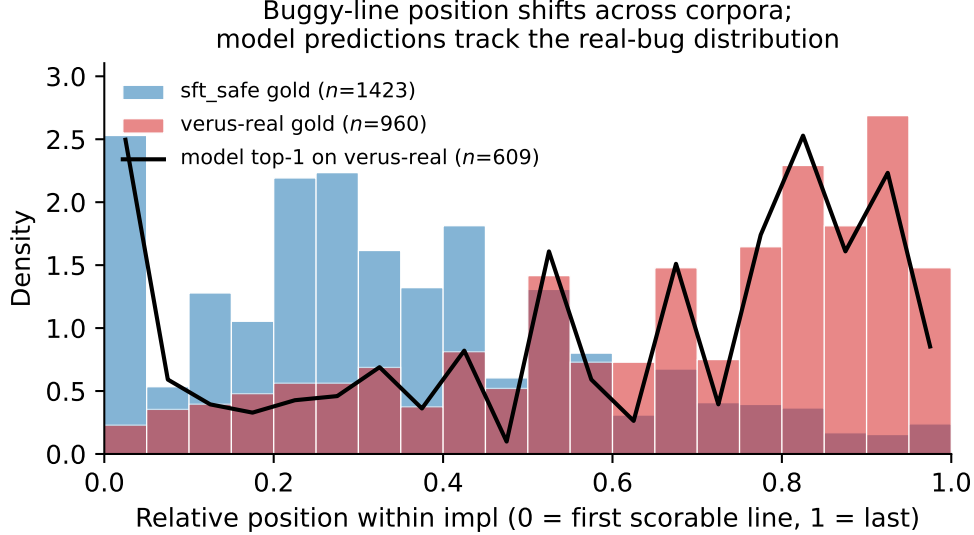


Figure 4: Buggy-line position distribution shifts between training and dev-test corpora (blue: `sft_safe` gold labels, median 0.31; red: `verus` dev-test gold labels, median 0.74). The black line shows our model’s top-1 predictions on dev-test failures (median 0.67) — tracking the dev-test distribution, not the training prior. $\text{Spearman}(\text{energy}, \text{position}) = +0.03$ per impl.

5 Discussion

What this paper claims. The headline is *calibration*, not specialist superiority. The scalar-head hybrid is a working per-line Verus fault localizer (top-3 = 0.84, AUROC = 0.78, Table 12), and the strip-FAILS audit pipeline is a transferable contribution. But Section 4.5 finds no closed-loop repair advantage over LLM-only or LLM-self-judged baselines. The paper’s claim is the calibration triple (Tables 1, 2 and 4) plus the audit pipeline, not a deployable specialist recommendation.

What this paper does not claim. The architecture is a discriminative EBM in the LeCun sense [4]; we do not claim generative density modeling.

Limitations.

1. **The Verus dev-test corpus is test-suite-authored, not production-derived.** Bugs are hand-written failing test cases from `verus-lang/verus` contributors and therefore reflect what Verus developers found worth testing (parser-edge cases, type mismatches, postcondition counter-examples), not bugs reported by production users. They are a substantially harder transfer target than the mutator-induced held set, but the gap from production-defect evaluation in the broader fault-localization literature should be kept in mind.
2. **The closed-loop CEGIS evaluation is underpowered.** We ran $n = 100$ tractable single-buggy-line impls. The three arms cluster at 25–30% repair-rate; A-vs-C has $p = 0.06$ but is not significant. A larger evaluation ($n \geq 500$) is needed.
3. LSE- T annealing during marker-leaky LSE training (artifact run #7) caused observed whole-impl AUROC oscillations as the effective aggregator changed mid-training — the “assessment drift” pathology of Zhang et al. [10]. The scalar-head hybrid sidesteps this by eliminating the LSE aggregator.
4. We did not run continued pretraining on public Verus code. A corpus audit of `verus-lang/verus` and `verus-lang/verus-tutorial` found only about 2.85MB of Rust files containing `verus!` macros, too small to justify a CPT stage under the hackathon compute budget. All domain adaptation in this paper

therefore comes from supervised LoRA and the localization losses, not from unsupervised Verus-code pretraining.

5. We evaluate on Verus only. Transfer to Dafny [5] or F* requires re-tokenization and is left unaddressed. A small HumanEvalPack-Rust probe [30] suggests out-of-Verus deployment would need retraining or a verifier-agnostic per-line head.
6. Single seed. We report bootstrap CIs over the held set but do not retrain across seeds. A LoRA-only re-init ensemble (Lakshminarayanan et al. 2017) over $k = 3$ seeds is the natural cheap variance estimate; we defer it to a longer revision.
7. Subgroup gap on **ensures**-clause failures (Table 6): top-3 drops to 0.59 on $n = 22$ postcondition bugs. The model’s per-line head can identify the line where a postcondition violation *surfaces* less reliably than where an **assert** or arithmetic statement fails, likely because the signal for an **ensures** violation lives across the entire impl rather than at one local site. A global-attention head alongside the local sentinel head is the natural architectural follow-up.

The required appendix “Limitations and Dual-Use Considerations” summarizes these risks in submission-checklist form and adds the dual-use analysis.

Implications for scalable oversight of AI-written code. With current frontier LLMs as proposer/rewriter, this small specialist is not the differentiating component in our CEGIS loop. That does not make attribution irrelevant; it shifts the question to what kind of signal helps repair, and which actor should produce it.

Future work. The most informative follow-ups are larger- n CEGIS, explicit self-judging-loop tests, OOD Verus features, ensemble routing, a global head for **ensures**-clause failures [31], and a seeded ensemble for variance estimation.

6 Conclusion

We built a 1.5B-parameter energy-based per-line scorer for Verus (AUROC = 0.78, top-1 = 0.56, top-3 = 0.84 on the marker-stripped dev-test corpus) and compared it with frontier LLMs on per-line ranking, whole-implementation discrimination, and closed-loop CEGIS repair. The specialist beats static baselines, but frontier LLMs match or beat it on all three comparisons; CEGIS repair-rate-at-1 is 0.25 specialist vs. 0.30 LLM-only and 0.30 LLM-self-judged at $n = 100$.

The durable contributions are the calibrated small-model baseline, the line-labeled Verus corpus, the scalar-head ablation, and the strip-FAILS audit pipeline. That audit invalidated an earlier marker-reliant result and revealed that our fix overshot into marker-aversion, while frontier LLMs are nearly marker-invariant. We release the corpus, model, checkpoints, audit scripts, LLM records, and CEGIS records. The clearest next questions are larger- n CEGIS, explicit self-judging-loop tests, OOD Verus features, and routing/ensemble approaches.

Code and Data

- Code repository: <https://github.com/ozlabsai/VericodingEBM>
- Live demo: <https://ozlabsai.github.io/VericodingEBM>
- Hugging Face model: <https://huggingface.co/ozlabs/VericodingEBM>
- Hugging Face dataset: <https://huggingface.co/datasets/ozlabs/VericodingEBM>
- Verus dev-test corpus: `artifacts/real_bugs/records.jsonl` (1,492 records, 609 FAIL-with-labels, derived from `verus-lang/verus` test suite at HEAD).

- Strip-FAILS audit pipeline: `scripts/strip_fails_reeval.py` (regenerates the audited corpus from the raw `// FAILS`-marked source for any model whose records use the same schema).
- Headline checkpoint (scalar-head hybrid; artifact run #10): `checkpoints/run10/`. Intermediate checkpoints (`run7_step500`, `run9`) are released alongside for reproduction of the audit narrative.

LLM Usage

We used Claude Opus to draft code, prose, and to dispatch literature-search sub-agents. All results, numbers, and claims were independently verified against logs and stored artifacts. The data parsers, model architecture, losses, training loop, eval scripts, baselines, Verus dev-test scraper, and analysis pipeline were co-authored with Claude and reviewed before commit.

References

- [1] A. Lattuada, T. Hance, C. Cho, M. Brun, I. Subasinghe, Y. Zhou, J. Howell, B. Parno, C. Hawblitzel. Verus: Verifying Rust Programs Using Linear Ghost Types. *OOPSLA 2023*.
- [2] Microsoft Research. *Verus Training Data* (HuggingFace dataset `microsoft/Verus_Training_Data`), 2024.
- [3] A. Solar-Lezama, L. Tancau, R. Bodík, S. Seshia, V. Saraswat. Combinatorial sketching for finite programs. *ASPLOS 2006*.
- [4] Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, F. Huang. A tutorial on energy-based learning. In *Predicting Structured Data*, MIT Press, 2006.
- [5] K. R. M. Leino. Dafny: An Automatic Program Verifier for Functional Correctness. *LPAR 2010*.
- [6] H. Lightman, V. Kosaraju, Y. Burda, H. Edwards, B. Baker, T. Lee, J. Leike, J. Schulman, I. Sutskever, K. Cobbe. Let’s Verify Step by Step. *arXiv:2305.20050*, 2023.
- [7] P. Wang, L. Li, Z. Shao, R. X. Xu, D. Dai, Y. Li, D. Chen, Y. Wu, Z. Sui. Math-Shepherd: Verify and Reinforce LLMs Step-by-Step without Human Annotations. *ACL 2024* (*arXiv:2312.08935*).
- [8] A. Setlur, C. Nagpal, A. Fisch, X. Geng, J. Eisenstein, R. Agarwal, A. Agarwal, J. Berant, A. Kumar. Rewarding Progress: Scaling Automated Process Verifiers for LLM Reasoning. *arXiv:2410.08146*, 2024.
- [9] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, J. Schulman. Training Verifiers to Solve Math Word Problems. *arXiv:2110.14168*, 2021.
- [10] Z. Zhang, C. Zheng, Y. Wu, B. Zhang, R. Lin, B. Yu, D. Liu, J. Zhou, J. Lin. The Lessons of Developing Process Reward Models in Mathematical Reasoning. *arXiv:2501.07301*, 2025.
- [11] W. Grathwohl, K.-C. Wang, J.-H. Jacobsen, D. Duvenaud, M. Norouzi, K. Swersky. Your Classifier is Secretly an Energy Based Model and You Should Treat it Like One. *ICLR 2020*.
- [12] Y. Du, I. Mordatch. Implicit Generation and Modeling with Energy Based Models. *NeurIPS 2019*.
- [13] A. Bakhtin, Y. Deng, S. Gross, M. Ott, M. Ranzato, A. Szlam. Residual Energy-Based Models for Text. *JMLR 2021* (*arXiv:2004.11714*).
- [14] R. Abreu, P. Zoetewij, A. J. C. van Gemund. On the Accuracy of Spectrum-based Fault Localization. *TAICPART-MUTATION 2007*.
- [15] R. Just, D. Jalali, L. Inozemtseva, M. D. Ernst, R. Holmes, G. Fraser. Are mutants a valid substitute for real faults in software testing? *FSE 2014*.
- [16] M. Pradel, K. Sen. DeepBugs: A Learning Approach to Name-based Bug Detection. *OOPSLA 2018*.
- [17] M. Allamanis, H. Jackson-Flux, M. Brockschmidt. Self-Supervised Bug Detection and Repair. *NeurIPS 2021*.
- [18] B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, T. Liu, J. Zhang, B. Yu, K. Lu, K. Dang, Y. Fan, Y. Zhang, A. Yang, R. Men, F. Huang, B. Zheng, Y. Miao, S. Quan, Y. Feng, X. Ren, X. Ren, J. Zhou, J. Lin. Qwen2.5-Coder Technical Report. *arXiv:2409.12186*, 2024.

- [19] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen. LoRA: Low-Rank Adaptation of Large Language Models. *arXiv:2106.09685*, 2021.
- [20] M. Renieris, S. P. Reiss. Fault Localization with Nearest Neighbor Queries. *18th IEEE International Conference on Automated Software Engineering (ASE 2003)*, pp. 30–39, 2003. DOI: 10.1109/ASE.2003.1240294.
- [21] A. Hindle, E. T. Barr, Z. Su, M. Gabel, P. Devanbu. On the Naturalness of Software. *34th International Conference on Software Engineering (ICSE 2012)*, pp. 837–847, 2012. DOI: 10.1109/ICSE.2012.6227135.
- [22] R.-M. Karampatsis, H. Babii, R. Robbes, C. Sutton, A. Janes. Big Code $\neq$ Big Vocabulary: Open-Vocabulary Models for Source Code. *42nd International Conference on Software Engineering (ICSE 2020)*, pp. 1073–1085, 2020. DOI: 10.1145/3377811.3380342.
- [23] Anthropic. Claude Opus 4.7 model card. <https://www.anthropic.com/claude/opus>, 2026. Accessed via OpenRouter slug `anthropic/claude-opus-4.7`.
- [24] OpenAI. GPT-5.5 system card. <https://openai.com/gpt-5.5>, 2026. Accessed via OpenRouter slug `openai/gpt-5.5`.
- [25] Google DeepMind. Gemini 3.5 Flash technical report. <https://deepmind.google/gemini-3.5>, 2026. Accessed via OpenRouter slug `google/gemini-3.5-flash`.
- [26] Qwen Team. Qwen 3.7 Max model card. <https://qwenlm.github.io/blog/qwen3.7-max/>, 2026. Accessed via OpenRouter slug `qwen/qwen3.7-max`.
- [27] DeepSeek AI. DeepSeek-V4 technical report. <https://github.com/deepseek-ai/DeepSeek-V4>, 2026. Accessed via OpenRouter slug `deepseek/deepseek-v4-pro`.
- [28] K. Zhang, Z. Li, J. Li, G. Li, Z. Jin. Self-Edit: Fault-Aware Code Editor for Code Generation. *ACL 2023*.
- [29] E. R. DeLong, D. M. DeLong, D. L. Clarke-Pearson. Comparing the areas under two or more correlated receiver operating characteristic curves: a nonparametric approach. *Biometrics* 44(3):837–845, 1988.
- [30] N. Muennighoff, Q. Liu, A. Zebaze, Q. Zheng, B. Hui, T. Y. Zhuo, S. Singh, X. Tang, L. von Werra, S. Longpre. OctoPack: Instruction Tuning Code Large Language Models. *ICLR 2024* (arXiv:2308.07124).
- [31] M. Ilse, J. Tomczak, M. Welling. Attention-based Deep Multiple Instance Learning. *ICML 2018*.
- [32] X. Sun, W. Xu. Fast Implementation of DeLong’s Algorithm for Comparing the Areas Under Correlated Receiver Operating Characteristic Curves. *IEEE Signal Processing Letters* 21(11):1389–1393, 2014. DOI: 10.1109/LSP.2014.2337313.
- [33] D. Kaushik, E. Hovy, Z. C. Lipton. Learning the Difference that Makes a Difference with Counterfactually-Augmented Data. *ICLR 2020*.
- [34] Y. Goyal, Z. Wu, J. Ernst, D. Batra, D. Parikh, S. Lee. Counterfactual Visual Explanations. *ICML 2019* (arXiv:1904.07451).
- [35] V. Veitch, A. D’Amour, S. Yadlowsky, J. Eisenstein. Counterfactual Invariance to Spurious Correlations in Text Classification. *NeurIPS 2021*.
- [36] M. R. I. Rabin, N. D. Q. Bui, K. Wang, Y. Yu, L. Jiang, M. A. Alipour. On the Generalizability of Neural Program Models with Respect to Semantic-Preserving Program Transformations. *Information and Software Technology*, 2021.
- [37] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, D. Krishnan. Supervised Contrastive Learning. *NeurIPS 2020* (arXiv:2004.11362).
- [38] T. Wang, P. Isola. Understanding Contrastive Representation Learning through Alignment and Uniformity on the Hypersphere. *ICML 2020* (arXiv:2005.10242).
- [39] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, H. Li. Learning to Rank: From Pairwise Approach to Listwise Approach (ListNet). *ICML 2007*.
- [40] F. Schroff, D. Kalenichenko, J. Philbin. FaceNet: A Unified Embedding for Face Recognition and Clustering. *CVPR 2015* (arXiv:1503.03832).
- [41] C.-Y. Wu, R. Manmatha, A. J. Smola, P. Krähenbühl. Sampling Matters in Deep Embedding Learning. *ICCV 2017* (arXiv:1706.07567).

- [42] F. Xia, T.-Y. Liu, J. Wang, W. Zhang, H. Li. Listwise Approach to Learning to Rank: Theory and Algorithm (ListMLE). *ICML 2008*.
- [43] T.-Y. Lin, P. Goyal, R. Girshick, K. He, P. Dollár. Focal Loss for Dense Object Detection. *ICCV 2017* (arXiv:1708.02002).

A Limitations and Dual-Use Considerations

Limitations. The main technical risks are false localization, benchmark mismatch, and scalability. False positives can waste repair budget on harmless lines; false negatives can hide the proof obligation or implementation line that actually needs attention. The Verus dev-test set is drawn from hand-authored test-suite failures, not deployed biosecurity or production-verification incidents, so edge cases such as multi-file proofs, library specifications, concurrency reasoning, solver timeouts, and postcondition failures may be underrepresented. The closed-loop CEGIS study is also small ($n = 100$) and single-line; it does not establish repair gains for multi-bug programs or long verification pipelines. Finally, the model is single-seed and Verus-specific; transfer to Dafny, F*, or ordinary Rust should be treated as unvalidated.

Future improvements. The highest-priority follow-ups are a larger CEGIS evaluation, multi-seed LoRA ensembles, multi-file Verus benchmarks, OOD tests on new Verus features, a global-attention head for nonlocal **ensures**-clause failures, and routing policies that decide when to trust the specialist versus a frontier LLM localizer. For biosecurity-relevant deployment, the localizer should be evaluated on representative access-control, screening, audit-log, and lab-automation verification tasks rather than only on upstream Verus tests.

Dual-use considerations. The intended use is defensive: helping researchers audit and repair formally verified code that protects high-consequence systems. The same localization signal could also reduce the cost of modifying verified systems, including attempts to bypass safety checks or weaken specifications. We therefore recommend releasing artifacts with clear provenance, benchmark-only examples, and verifier logs; using the tool as an advisory signal rather than an automatic patch acceptor; requiring human review for security- or biosecurity-relevant code; and tracking whether the model points to specification lines whose weakening would make unsafe behavior easier rather than harder to prevent.

B Additional results

Eval-time aggregator sweep on Verus dev-test records. Same per-line energies, different reducer: mean=0.471, max=0.484, min=0.490, last=0.517, sum=0.530, softplus-sum=0.475, LSE(train-time)=0.487 (Table 5 in main text).

Z-score top- k aggregator (Option B' from [4]-style reformulation; not headline-defensible due to “assessment drift”): within-impl z-score normalize per-line energies, then mean of top- k . On Verus dev-test records: $k=1$: 0.504; $k=2$: 0.451; $k=3$: 0.424; $k=5$: 0.420. No reducer escapes the chance band; the ceiling holds.

C Significance tests: scalar-head hybrid vs. baselines

Table 11 gives paired significance tests for the scalar-head hybrid (stripped) against the six static baselines on the Verus dev-test corpus. Top- k tests are paired McNemar (exact binomial when $b_{01} + b_{10} < 25$, χ^2 with continuity correction otherwise). AUROC tests are paired DeLong following Sun and Xu’s fast implementation [32] of the original procedure [29]. All tests use the same $n = 1,492$ records (or $n_{\text{labeled}} = 609$ FAIL records for top- k) joined by `impl_id`; no records are excluded.

The marker-keyword baseline ($p = 0.006$ on AUROC) shows the smallest gap because by design it is a near-oracle for line-level localization on the unstripped corpus—but even there, the scalar-head hybrid’s

Table 11: Paired significance tests: scalar-head hybrid (stripped) vs baselines on $n = 1,492$ Verus dev-test records ($n_{\text{labeled}} = 609$ FAIL with line labels). Top- k p-values are McNemar (paired binomial); AUROC p-values are DeLong. **Bold** = scalar-head hybrid better.

Baseline	n	top-1 rate	p	top-3 rate	p	AUROC val	p
random	609	0.105 vs 0.560	$< 10^{-4}$	0.366 vs 0.839	$< 10^{-4}$	0.405 vs 0.778	$< 10^{-4}$
length	609	0.328 vs 0.560	$< 10^{-4}$	0.668 vs 0.839	$< 10^{-4}$	0.463 vs 0.778	$< 10^{-4}$
keyword verus	609	0.544 vs 0.560	0.529	0.767 vs 0.839	$< 10^{-4}$	0.373 vs 0.778	$< 10^{-4}$
keyword marker	609	0.997 vs 0.560	$< 10^{-4}$	1.000 vs 0.839	$< 10^{-4}$	0.673 vs 0.778	0.006
qwen surprisal	609	0.002 vs 0.560	$< 10^{-4}$	0.141 vs 0.839	$< 10^{-4}$	0.597 vs 0.778	$< 10^{-4}$
diff sibling	609	0.072 vs 0.560	$< 10^{-4}$	0.250 vs 0.839	$< 10^{-4}$	0.476 vs 0.778	$< 10^{-4}$

impl-level discrimination is statistically meaningfully better, and on the stripped corpus (Section 4.2) the gap widens further.

D Reproducibility

All numbers in this paper tie to artifacts saved on disk in the `artifacts/` directory of our released codebase:

- **Verus dev-test corpus:** `artifacts/real_bugs/records.jsonl` (1,492 records, 609 line-labeled FAIL examples). Generated by `scripts/scrape_verus_real_bugs.py` from the `verus-lang/verus` GitHub repository’s test suite at HEAD (source/`rust_verify_test/tests/*.rs`, 132 files).
- **Model scoring:** run `scripts/run_mutator_probe.sh` on `checkpoints/<run>` to produce `artifacts/real_bugs/<run>/eval_records.jsonl` for any released checkpoint (`run7_step500`, `run9`, `run10`).
- **Three-way audit:** run `scripts/strip_fails_reeval.py` and `scripts/token_mask_reeval.py`. These produce the `no_surgery`, `stripped`, and `token_masked` arms used in Tables 3 and 8.
- **Significance + subgroups:** run `scripts/significance_and_subgroups.py` on the above eval records.
- **Position-shift figure:** `scripts/figure_position_shift.py`.
- **LLM baselines:** `scripts/baseline_llm_zeroshot.py` (ranking-prompt) and `scripts/baseline_llm_discrimination.py` (discrimination-prompt). OpenRouter slug names listed in the reproducibility note in Section 4.3.
- **CEGIS three-arm experiment:** run `scripts/cegis_demo_3arm.py` and `scripts/cegis_rerun_arm_b.py`. These produce `artifacts/cegis/3arm_results_v2.jsonl` and `summary_3arm.json`. Requires a local Verus toolchain binary; we used the prebuilt `verus-0.2026.05.17.e479cce-arm64-macos`.

Hyperparameters. LoRA rank 16, alpha 32, dropout 0.05; all-linear target modules plus rank-8 `embed_tokens`. Per-line head: MLP $1536 \rightarrow 256 \rightarrow 1$, GELU, dropout 0.1, init-std 0.02. Optimizer AdamW ($\beta_1=0.9$, $\beta_2=0.95$), peak LR 1×10^{-4} , cosine schedule with 3% warmup decaying to 10% of peak. LSE temperature linearly annealed $1.0 \rightarrow 0.3$ across the full schedule (the source of the assessment-drift event documented in Section 5). Loss weights: $\lambda_{\text{spec}} = 3.0$, $\lambda_{\text{line}} = 1.0$, line-loss margin 1.0. Per-batch composition: at least 2 trajectory triples + 4 `sft_safe` pairs. Micro-batch 16, no grad-accum, bf16 precision, gradient checkpointing on. Max sequence length 4096. Sentinel token: `<|fim_pad|>` (id 151662 in Qwen2.5-Coder tokenizer).

Hardware. Single A100 SXM 80GB (RunPod community cloud, USD \$1.39–1.49/hr). Marker-leaky LSE (run #7) total wallclock: $\sim 3\text{h}$ to reach step 1845 before stop-point. The step-500 checkpoint we report from required ~ 30 minutes of training.

E Marker-leak fix: from mixed-marker augmentation to scalar-head hybrid

The strip-FAILS audit (Section 4.8) revealed a Qwen pretraining-prior leak through `// FAILS` comment tokens. The training corpus contains *zero* such markers, while 42.9% of dev-test records do; the model inherited the marker→buggy association from pretraining. We attempted three fixes.

Mixed-marker augmentation (run #8; failed). We injected markers on any line and added positive distractors (`// ok// verified`), following counterfactual augmentation work [33, 34, 35, 36]. Held-set performance collapsed (AUROC 0.53, top-3 0.23), likely because distractors on buggy lines created within-class label noise for the contrastive objective [37, 38].

Adversarial-marker LSE (run #9; partial fix). We then placed `// FAILS`-family markers only on random non-buggy lines. This shrank the marker-leak top-3 gap from 33pp to 4pp, but whole-impl AUROC inverted to 0.40 while top-3 reached 0.87. The LSE aggregator inherited an implementation-level bias that simple mean-subtraction did not fix.

Scalar-head hybrid (run #10; architectural fix). We therefore added a directly trained scalar attention-pool head over [spec; impl] hidden states, avoiding LSE aggregation, with losses $L = \lambda_{\text{scalar}} L_{\text{impl-pair}} + \lambda_{\text{list}} L_{\text{listwise}} + \lambda_{\text{pair}} L_{\text{semi-hard}}$ where L_{listwise} is ListNet softmax cross-entropy within an impl [39] and $L_{\text{semi-hard}}$ is per-line pairwise hinge with FaceNet-style semi-hard top- k mining [40, 41]. The impl-pair loss is logistic pairwise softplus($E^+ - E^-$).

Table 12: Scalar-head hybrid vs. baselines on the Verus dev-test corpus ($n=1,492$, 609 FAIL with labels). Bootstrap 95% CIs from 500 resamples. **Stripped is the canonical eval** (marker-free).

	top-1	top-3	AUROC
Leaky LSE (run #7), with markers	0.73	0.93	0.49
Leaky LSE (run #7), stripped	0.44	0.60	0.44
Adv-marker LSE (run #9), token-masked	0.45	0.80	0.51
Adv-marker LSE (run #9), stripped	0.45	0.76	0.49
Scalar-head hybrid (run #10), token-masked	0.064	0.388	0.782
Scalar-head hybrid (run #10), stripped	0.560	0.839	0.778
Scalar-head hybrid (run #10), no-surgery	0.038	0.238	0.842

Scalar-head hybrid three-way evaluation. Reading. The scalar-head hybrid is the first checkpoint in the sweep where AUROC and top-3 are both above chance. Its stripped result is the canonical leak-free metric. With markers retained, however, top-1 collapses to 0.038–0.064: adversarial marker training over-corrected into marker-aversion, so the with-marker arm is reported only for transparency.

Ablation: what fixed AUROC. We trained four single-component ablations: no scalar head (A1), no listwise loss (A2), no semi-hard mining (A3), and hinge instead of logistic impl-pair loss (A4). All use the same data and dev-test scorer.

Reading. Removing the scalar head collapses AUROC from 0.78 to 0.46, making it the load-bearing change. Removing listwise loss or semi-hard mining hurts AUROC but less dramatically; switching logistic back to hinge is within noise. Top- k is comparatively stable because the per-line head remains present.

F Strip-FAILS re-evaluation (audit pipeline)

The audit table and its discussion are in Section 4.8 (Table 8). This appendix records the released code needed to rerun it:

Table 13: Single-knob ablations of the scalar-head hybrid on the marker-stripped Verus dev-test corpus ($n=1,492$). Attribution order: **scalar head** $\gg$ **listwise** $>$ **semi-hard mining** $\gg$ **logistic loss**. The scalar attention-pool head is the load-bearing architectural change; removing the listwise or semi-hard component degrades AUROC substantially but leaves top- k recall comparatively intact; the logistic-vs-hinge choice is within-noise.

	AUROC	top-1	top-3
Scalar-head hybrid (run #10; reference)	0.778	0.560	0.839
No scalar head (A1; LSE)	0.455	0.516	0.819
No listwise (A2)	0.592	0.598	0.836
No semi-hard mining (A3)	0.663	0.547	0.790
Hinge instead of logistic (A4)	0.784	0.522	0.847

- `scripts/strip_fails_reeval.py` — reads `records.jsonl`, strips the `// FAILS` substring (and any trailing characters) from each line, re-tokenizes, re-scores with the named checkpoint, and writes `stripped.jsonl`.
- Three-way harness: `no_surgery`, `stripped`, and `token_masked`; the token-masked arm checks whether the effect is tied to marker tokens rather than nearby context.
- Mitigation: `configs/run9_embed_surgery.yaml` and `configs/run10_hybrid.yaml` demonstrate the adversarial-marker-injection training fix and the architectural fix; both shrink the with-marker-vs-stripped top-3 gap from 33pp to ≤ 4 pp.

We recommend running the three-way harness before reporting transfer numbers on comment-marked Verus tests.

Single-seed caveat. The audit numbers in Table 8 are from a single training seed. Bootstrap CIs capture sampling variance, not model-seed variance; a seeded LoRA ensemble is left for follow-up.