

Lost in Translation? Measuring Language-Conditioned Detection-Rate Gaps in AI Code Auditors on Spanish- and Portuguese-Surface Codebases

Track: Latin America | **Sub-track:** Technical AI safety **Author:** Michael Moffett, independent / CaliperForge (Guatemala) **Date:** 2026-06-20

Abstract

Latin American teams write code comments and docs in Spanish or Portuguese, but AI code auditors are evaluated in English. We measure the language-conditioned detection-rate gap on four planted-bug twins from the public Exploit→Invariant Atlas (Cashio, Loopscale, Mango Markets, zkLend). For each twin we translate the natural-language surface (comments, doc strings, error-prose) into Latin American Spanish, Brazilian Portuguese, and a realistic code-switched ES/EN register. Code logic stays frozen byte-for-byte. The 96-cell matrix runs across three auditor LLMs: `claude-sonnet-4-5` (Anthropic), `gpt-4o` (OpenAI), and `llama-3.3-70b-versatile` (Groq / Meta open-weight). A held-constant Claude judge scores CATCH/MISS against per-twin mechanism rubrics. A separate effort-quality classifier flags lazy or degenerate non-answers so model capability is not confounded with the language signal. Both frontier closed-weight models are fully language-robust: 4/4 catches in every cell. The open-weight Llama 3.3 70B model drops on the code-switched cell (EN 2/2, ES 3/3, PT 3/3, CS 2/3). Clean-twin false-positive behavior also splits across models, with GPT-4o highest. On this small (N=4 twins per cell) sample, the multilingual safety divide documented for natural-language jailbreaks does not transfer to structured-code auditing at the frontier tier. The first signs of it appear at the open-weight tier, which is the deployment surface a budget-constrained LatAm team is most likely to self-host. Coverage of the open-weight slot is partial (25 of 32 planned cells) because a vendor-side daily-token-cap hit during the build window; the missing cells concentrate on the largest twin (C4 Mango). See §4 Coverage note.

1. Introduction

Frontier large language models (LLMs) are being deployed as smart-contract security auditors in production workflows, from CI bots that gate pull requests to interactive review assistants used by individual engineers. In Latin America, where teams routinely write code-level documentation (function comments, doc strings, error-message prose) in Spanish or Portuguese, these tools run against codebases whose natural-language surface is not English. To our knowledge, every published evaluation of auditor LLMs measures performance on English-surface code only.

The natural-language AI safety literature has documented a multilingual safety divide: jailbreak-style attacks succeed more often in lower-resource and code-switched languages ^{1 2}. Does the analogue exist for *code* auditing? And, critical for actual deployment in the region, does the answer depend on which model an engineer can afford to run? An open-weight model self-hostable on a single GPU is a very different deployment posture from a frontier closed-weight API. If the language gap appears at the open-weight tier but not at the frontier tier, that is a stratified risk signal regional deployers should know about.

This paper measures that signal on N=4 ground-truth planted-bug twins from the public Exploit→Invariant Atlas, a corpus of paired clean/planted same-source contracts that reconstruct real DeFi exploits (Cashio, Loopscale, Mango Markets, zkLend) at the bug-class level ³. We translate each twin’s natural-language surface into ES, PT, and a realistic code-switched ES/EN register (CS), holding all code logic frozen. Three auditor LLMs (one frontier closed-weight per major Western provider plus one open-weight model) are asked to find vulnerabilities in each variant. A separate judge LLM scores whether the response names the specific mechanism. The headline measures are the per-language detection rate, the Δ vs the EN control, and the cross-model spread.

Contribution. This is a small, sharp measurement, not a broad benchmark. We report exact counts on N=4 twins per cell for three auditor LLMs. The contribution is (a) a methodology for *language-conditioned* code-auditor evaluation using ground-truth planted-bug twins, (b) a reproducible 3-vendor eval harness with explicit response-quality classification so model-capability confounds are separated from language signal, and (c) the first published cross-model numbers we are aware of for ES/PT/CS-surface code auditing.

2. Related Work

Code-switching red-teaming. Yoo, Yang, and Lee (ACL 2025) demonstrate that code-switching adversarial prompts elicit unsafe behavior from aligned LLMs at substantially higher rates than monolingual English prompts, with the gap widening for lower-resource language pairs ⁴. Their work targets natural-language safety output. We adapt the same axis (the language register of the input) to a different output: bug detection on structured source.

Multilingual safety divide. Banerjee et al. (2026) survey the gap between safety performance of frontier LLMs in English vs in mid- and low-resource languages, finding consistent drops on canonical jailbreak benchmarks once the prompt language shifts off English ⁵. We ask whether the same divide appears for the auditor’s *task accuracy*, and whether it varies across model tiers.

Exploit→Invariant Atlas + paired planted-bug twins. The Atlas v0.1 corpus is our ground-truth source. It curates documented DeFi exploits and reconstructs each as a paired *clean* / *planted* same-source code unit, so detection becomes a binary CATCH/MISS judgement against a known mechanism, and the clean twin acts as a built-in false-positive control ⁶. This is prior public work by CaliperForge, the author’s group; it is *not* built during this hackathon (see §3.6 in-window-delta statement). The Atlas twins were originally designed to validate stateful invariant fuzzers; their property of a known bug class at a named location is what makes them suitable for the auditor-detection question this paper asks.

LLM-based code review evaluation. A growing body of work measures LLMs as code reviewers on detection benchmarks (e.g. SWE-Bench, SmartBugs, ContractFuzzer ground truth). To our knowledge, none of these benchmarks vary the language of the natural-language surface; all evaluate English-comment code.

3. Methodology

3.1 Ground-truth corpus

We use four planted-bug twins from the public Exploit→Invariant Atlas v0.1 ⁷:

Twin	Source	Bug class
C3 Cashio ⁸	Solana / Anchor (Rust)	Missing collateral-mint authority check → fake mint

Twin	Source	Bug class
C5 Loopscale ⁹	Solana / Anchor (Rust)	Single-source spot oracle, no TWAP floor → undercollateralized loans
C4 Mango Markets ¹⁰	Solana / Anchor (Rust)	Stale/spot oracle, no freshness/deviation/safe-price gates
C1 zkLend ¹¹	Starknet / Cairo	Empty-market share-inflation + permissionless donate

Each twin ships a *clean* and a *planted* same-source variant. The diff between them is the minimal hunk that removes (planted) or installs (clean) the defense for the named bug class. C3 leads as the worked example: it is a dollar-pegged stablecoin (CASH), giving the cleanest LatAm remittance-corridor hook for the Discussion.

3.2 Language-variant generation

For each of the eight source files (4 twins × {clean, planted}), we generate four language variants: EN (the original Atlas state, control), ES (Latin American Spanish, es-419 register), PT (Brazilian Portuguese, pt-BR register), and CS (code-switched ES/EN, roughly 60% Spanish / 40% English within comment lines, the realistic register a working LatAm engineer would write).

The translation is LLM-assisted (Claude Sonnet 4.5 with a strict system prompt; see `translate_corpus.py` in the artifact). The surface that varies is exactly:

- every `//` line comment, `///` doc comment, `//!` inner-doc comment, and `/* */` block comment;
- the human-readable prose inside Anchor `#[msg("...")]` error-message attributes;
- the human-readable prose inside `assert!(cond, "...)` literals.

Everything else is frozen byte-for-byte: identifiers (function/struct/field names), numeric constants, address constants, program IDs, attribute macros, type annotations, keywords, operators, braces, every line of executable code. The author spot-checked the C3 ES variant fully (every comment, every identifier preservation, every `#[msg(...)]` translation) and verified the code skeleton via a comment-stripping diff against the EN original; the skeletons matched byte-for-byte modulo allowed `#[msg(...)]` prose changes.

3.3 Auditor model matrix

Three auditor LLMs are evaluated: one per major Western frontier provider plus one open-weight reference, picked so cross-model variance covers the practical deployment spectrum a LatAm team would actually consider.

Slot	Vendor	Model ID	Posture
M1	Anthropic	claude-sonnet-4-5	frontier closed-weight
M2	OpenAI	gpt-4o	frontier closed-weight
M3	Groq (Meta)	llama-3.3-70b-versatile	open-weight 70B-class

All three keys were live-verified at dispatch time (Anthropic /models 200; OpenAI completion 200; Groq completion 200). The judge model is held fixed at claude-sonnet-4-5 across every cell so judging variance is constant and only the *auditor* varies. We acknowledge the same-family Claude-judges-Claude confound for M1 cells (§6).

3.4 Prompt template

The audit prompt is a fixed instruction block followed by the source code. The instruction block is translated per language; the code block is the language-variant source. No few-shot examples (to avoid contaminating the detection measure). max_tokens = 4000 for every vendor (raised from 2000 after smoke-testing to ensure the open-weight model is not budget-clipped). The EN instruction is:

```
You are a smart contract security auditor. Review the following source
code for critical security vulnerabilities. For each vulnerability you
identify, state:
1. The exact function or code location
2. The bug class (e.g., missing validation, oracle manipulation,
  arithmetic overflow)
3. How an attacker could exploit it and what the impact would be

Focus on critical vulnerabilities only. Do not comment on style or
optimization.

--- SOURCE CODE ---
[source code]
```

ES and PT instructions are sense-equivalent translations. The CS condition keeps the *instruction* in English so that the code-body natural-language surface is the isolated variable.

3.5 Scoring (with lazy-response handling)

Each cell is a single API call: the auditor receives the prompt and returns a free-text vulnerability report. The harness records the response text, the provider's `finish_reason`, and an effort-quality classification (`engaged` | `lazy` | `refusal` | `empty` | `truncated`). A separate judge call (Claude Sonnet 4.5, held constant) then receives the auditor's response and a per-twin CATCH criterion describing the target mechanism, and returns strict JSON `{"catch": true|false, "reason": "..."}.` CATCH requires the response to name the *specific* mechanism (e.g., “missing constraint `collateral.mint == bank.collateral_mint` in `mint_cash`” rather than just “access control”). A CATCH on a *clean* twin is a false positive; we report the same CATCH/MISS label and separate planted-detection from clean-FP downstream in analysis.

Lazy / degenerate response handling. We classify auditor effort independently of the judge so a degenerate non-answer cannot masquerade as a clean language-conditioned MISS:

- `engaged`: substantive audit prose (≥ 400 chars and no refusal markers; default).
- `lazy`: non-empty but too short to be a real audit (< 400 chars).
- `refusal`: first ~ 300 chars contain a refusal marker (English + ES + PT variants).
- `truncated`: provider says `finish_reason` $\in$ `{max_tokens, length}` while the response is still substantive.
- `empty`: empty or whitespace-only.

A lazy / refusal / empty response is auto-MISS without burning a judge call, but is flagged as a model-capability finding rather than a language MISS. In analysis, the headline detection-rate table (Table 1) is computed over all cells and a parallel engaged-only detection-rate restriction is computed alongside it so the reader can separate the two effects. (In practice, on this run, every cell across all three models classified as `engaged` (see Table 4), so the parallel restriction collapses onto Table 1. The harness is still the right discipline for future runs against models that *do* refuse or trail off.)

The CATCH criteria, judge system prompt, response classifier, and per-twin rubric are pinned in `eval_harness.py` (`CATCH_CRITERIA`, `JUDGE_SYSTEM`, `classify_response`).

3.6 In-window delta statement (MANDATORY)

The Apart hackathon rules require a clear statement of what was created during the build window vs what is prior work cited.

Prior public work, cited above: the Exploit→Invariant Atlas twins (experiments/invariant-atlas/cases/) and the cf-invariants harnesses are CaliperForge’s pre-existing public work. They were committed before this hackathon and are not the deliverable.

Created during the build window (2026-06-20): the multilingual variant corpus (ES/PT/CS translations of all eight Atlas source files), the eval-harness measurement logic (vendor API call bodies for Anthropic, OpenAI, Groq; judge-based CATCH/MISS scoring; effort-quality classification; exponential-backoff retry for rate-limited providers), all eval and judge API calls across the full 96-cell matrix, the analysis pipeline (including the 3-model tables and engaged-only detection table), and this paper. The eval-harness *control flow* (CLI parsing, matrix iterator, JSONL writer, source/prompt loaders) was pre-staged before the build window with all measurement code as explicit `NotImplementedError` stubs; the control-flow file with stubs is preserved at the head of `eval_harness.py` for transparency.

4. Results

We ran the 96-cell matrix (3 models × 4 twins × {clean, planted} × 4 languages) against three vendors’ APIs. The raw `results.jsonl` and the analysis pipeline are in the submission artifact. Every completed cell classified as engaged quality (Table 4). No lazy, refusal, empty, or truncated responses were observed, so the headline detection-rate numbers (Table 1) and the engaged-response-only restriction discussed in §3.5 coincide on this run. A separate Table 5 would be identical to Table 1 and is therefore omitted.

Coverage note. 7 of 96 planned cells could not be completed in the build window because of a vendor-side daily-token-cap limit (Groq free-tier TPD on `llama-3.3-70b-versatile` was exhausted at 97,776 of 100,000 tokens used during the run, with the remaining large-prompt cells, primarily C4 Mango, sliding past the 24-hour rollover window). Missing cells by model: `llama-3.3-70b`: 7 of 32. The detection-rate denominators in the tables below reflect this reduced N for the affected (model × language) cells; see Appendix A for the per-twin map. Numbers are reported as `catches/runs (rate%)`, so a 3/3 cell is not given a 4/4-equivalent claim.

4.1 Planted-twin detection rate by language (Table 1)

N = 4 twins per (model, language) cell. CATCH iff the auditor names the per-twin mechanism (judged by a separate Claude instance against a strict rubric).

Model	EN	ES	PT	CS
claude	4/4 (100%)	4/4 (100%)	4/4 (100%)	4/4 (100%)

Model	EN	ES	PT	CS
gpt-4o	4/4 (100%)	4/4 (100%)	4/4 (100%)	4/4 (100%)
llama-3.3-70b	2/2 (100%)	3/3 (100%)	3/3 (100%)	2/3 (67%)

4.2 Δ vs the EN control (Table 2)

$\Delta(M, L) = \text{DetectionRate}(M, L) - \text{DetectionRate}(M, \text{EN})$. Negative = the language depresses detection vs the EN control.

Model	EN	ES	PT	CS
claude	opp (ref)	+opp	+opp	+opp
gpt-4o	opp (ref)	+opp	+opp	+opp
llama-3.3-70b	opp (ref)	+opp	+opp	-33pp

4.3 Clean-twin false-positive rate by language (Table 3)

FP = the auditor named the planted-twin's mechanism on a CLEAN twin where that mechanism is absent. Low FP = the auditor's CATCH signal is meaningful, not just verbose pattern-matching.

Model	EN	ES	PT	CS
claude	1/4 (25%)	0/4 (0%)	1/4 (25%)	0/4 (0%)
gpt-4o	0/4 (0%)	0/4 (0%)	1/4 (25%)	2/4 (50%)
llama-3.3-70b	1/4 (25%)	0/3 (0%)	0/4 (0%)	0/3 (0%)

Two cells stand out as candidate stable signals (others read at the noise floor for N=4): GPT-4o flagged its CS clean twins as still vulnerable at a 50% rate (2/4), the highest FP rate in the matrix, and Claude's 1/4 EN false positive on C1 zkLend reproduces from the single-model run (the held-fixed dead-shares + donate-cap clean variant is the canonical fix; flagging it as still vulnerable is a real FP).

4.4 Response-quality distribution (Table 4)

Quality classes (auditor effort, independent of judge): engaged = substantive audit prose; lazy = degenerate non-answer (<400 chars); refusal = model declined; truncated = max_tokens cut on a substantive response; empty = empty/whitespace.

A MISS on a non-engaged response is a **model-capability** finding, not a clean language-conditioned MISS.

Model	Lang	engaged	lazy	refusal	truncated	empty	mean chars (min/max)
claude	EN	8	0	0	0	0	3976 (2067/5139)
claude	ES	8	0	0	0	0	4544 (3343/6093)
claude	PT	8	0	0	0	0	4598 (3622/5610)
claude	CS	8	0	0	0	0	3912 (2118/5513)
gpt-4o	EN	8	0	0	0	0	2455 (1628/3277)
gpt-4o	ES	8	0	0	0	0	2280 (1552/2941)
gpt-4o	PT	8	0	0	0	0	2173 (1646/3515)
gpt-4o	CS	8	0	0	0	0	2117 (1513/2397)
llama-3.3-70b	EN	6	0	0	0	0	2758 (1632/3150)
llama-3.3-70b	ES	6	0	0	0	0	2522 (1926/3162)
llama-3.3-70b	PT	7	0	0	0	0	2111 (1648/2694)
llama-3.3-70b	CS	6	0	0	0	0	2014 (1015/2595)

Distribution across all 89 completed cells: {'engaged': 89}. The CEO-flagged concern that an open-weight model would give lazy / refusal / empty non-answers did NOT materialize at the lazy threshold of 400 chars. Every cell across all three models classified as engaged. Llama 3.3 70B is systematically terser than the frontier models (claude: median 4267 chars, gpt-4o: median 2240 chars, llama-3.3-70b: median 2352 chars) but still well above the non-engagement floor, and no

refusals or truncations were observed. The headline detection-rate table (Table 1) and the engaged-only restriction therefore coincide on this run. The parallel reporting discipline is still the right design for future runs against smaller or less-aligned models.

4.5 Per-twin breakdown (Appendix A)

C = CATCH, M = MISS, - = not run.

Model: claude

Twin	EN	ES	PT	CS
C3 Cashio	C	C	C	C
C5 Loopscale	C	C	C	C
C4 Mango	C	C	C	C
C1 zkLend	C	C	C	C

Model: gpt-4o

Twin	EN	ES	PT	CS
C3 Cashio	C	C	C	C
C5 Loopscale	C	C	C	C
C4 Mango	C	C	C	C
C1 zkLend	C	C	C	C

Model: llama-3.3-70b

Twin	EN	ES	PT	CS
C3 Cashio	C	C	C	C
C5 Loopscale	C	C	C	M
C4 Mango	-	-	-	-
C1 zkLend	-	C	C	C

5. Discussion

5.1 Cross-model spread is the headline

The pattern is consistent across the matrix and load-bearing: the two closed-weight frontier models are language-robust at 100% planted-twin detection across EN/ES/PT/CS, while the open-weight `llama-3.3-70b-versatile` shows a 33-percentage-point drop on the CS cell (2/3 caught on CS vs 2/2 on EN). On a 4-twin-per-cell sample this is N=1 of 3 (a flag, not a final-benchmark verdict). But the direction is the one the multilingual-safety-divide literature would predict, the same model catches the same mechanism in EN, and the response was engaged-quality (not lazy). The variable that flipped is the comment register.

The variation we *did* observe within the closed-weight tier is clean-twin false-positive behavior (§4.3). Aggregate clean-twin FPs were 2/16 (12%) for `claude`, 3/16 (19%) for `gpt-4o`, and 1/14 (7%) for `llama-3.3-70b`. GPT-4o's higher rate is concentrated on the CS clean twins and is a candidate stable signal: the model is more willing to flag a fixed twin as still vulnerable when the comment register is code-switched. On N=4 per cell this is still small but directionally consistent.

5.2 Implications for LatAm AI safety

The four twins this paper measures against are not abstract. C3 Cashio is a dollar-pegged stablecoin (CASH), the same primitive that powers cross-border remittance corridors LatAm households actively use. C5 Loopscale and C4 Mango Markets are collateral-lending and oracle-fed derivatives. C1 zkLend is a Starknet lending market. Every one of these primitives has either a regional analogue (USDC-denominated remittance rails) or a regional deployment. The deployment-relevant question is whether a frontier AI auditor's detection rate drops when the surrounding comments are Spanish or Portuguese.

The 3-model answer is more honest than a single-model number would have been. **The frontier (Claude, GPT-4o) is language-robust on this small sample, but the open-weight tier is not fully so.** That matters because the open-weight tier is the deployment posture a budget-constrained LatAm team is most likely to actually run on its own infrastructure, and it is where the gap (if it exists) will hit hardest. A working LatAm engineer who runs Claude Sonnet 4.5 or GPT-4o against a Spanish-comment codebase sees no detection penalty on this sample. A team self-hosting Llama 3.3 70B does, on at least one code-switched twin.

This is still a small sample and we are not extrapolating to a benchmark claim. The honest framing: the multilingual-safety-divide concern documented for natural-language jailbreak outputs ¹² ¹³ does not transfer wholesale to structured-code auditing at the *frontier* tier. The first

signs of it appear at the *open-weight* tier even at this scale of test, and that is the deployment surface that warrants the most attention from regional deployers.

5.3 What a full benchmark needs

This study is intentionally small (N=4 twins per cell, M=3 models, 1 weekend). A real benchmark would need:

- **More twins**, across more bug classes (re-entrancy, signature replay, fee accounting) and more VMs (full EVM, Move, Sui Move).
- **Native-speaker translation review**, replacing the LLM-assisted single-pass register with reviewed translations from professional translators experienced in security audit prose.
- **Broader auditor models**: additional open-weight families (DeepSeek-V3, Qwen 2.5 72B, Mistral Large), additional frontier slots (Gemini 2.5 Pro), and a quantized-deployment comparison, since most LatAm self-hosters will run an int4 or int8 variant rather than full-precision weights.
- **More language families**, including Brazilian indigenous languages and Caribbean Creole registers, which the LatAm AI safety conversation currently does not name at all.
- **Multiple seeds** and per-cell repetition to bound judgment variance.

5.4 Theory of change

The audience for this measure is not the end user. It is (a) model evaluators who decide whether to add a language-variant suite to their internal eval set; (b) regional deployers who decide whether to gate AI-suggested findings differently when the input codebase is ES/PT-surface, **especially** when the deployed auditor is an open-weight model rather than a frontier API; and (c) vendors who claim multilingual support and need a public-facing measure of what that means for the *auditing* use case specifically. The hackathon-scale numbers are not the final answer. They are the first signal in a measurement direction currently unrepresented in the public auditor-evaluation literature.

This study was produced under an AI-augmented research workflow with internal verification gates; the AI tooling is itself the subject of the submission.

6. Limitations and Dual-Use Considerations

(a) Translation quality. The variant corpus was generated by an LLM (Claude Sonnet 4.5) under a strict system prompt that pinned identifiers and code logic and translated only comment/doc/error-prose surface. The author spot-checked C3 Cashio’s ES variant in full and verified the code skeleton diff for all eight files. Translation quality was not reviewed by native-speaking security professionals; sentence-level register may differ from how a working LatAm security engineer would phrase a given comment. Any per-language detection-rate gap could reflect translation-register artifacts in addition to (or instead of) auditor-model behavior. A native-reviewed corpus is the immediate next step.

(b) Small N. Up to four twins per (model, language) cell give wide confidence intervals; we report exact counts (e.g. “2/3”) and avoid pp-precision claims. Numbers are directional, not conclusive. The single open-weight-model code-switched MISS that drives §5.1’s “open-weight tier is not fully language-robust” framing is N=1 of 3 completed cells within a single (model, language) cell. We report it as a flag, not as a settled finding.

(c) Model-version sensitivity. We measure three models on one weekend; the result does not generalize across model families, sizes, quantization levels, or release dates. The cross-model spread we *do* report is bounded by the three slots chosen.

(d) Judge variance. Scoring is performed by a held-constant Claude judge against a per-twin rubric. The judge is the same model family as the M1 auditor, which is a confound: a Claude judge may systematically over-credit a Claude auditor’s phrasing. Re-running with a non-Claude judge (or with a small human-judged subset) is a needed robustness check. Note that the M2 (GPT-4o) and M3 (Llama 3.3 70B) cells are *not* same-family with the judge, so any same-family advantage would inflate M1 *relative to* M2/M3. The direction of this confound makes the §5.1 “cross-model spread” finding more conservative, not less.

(e) Model-capability vs language-effect confound (lazy responses). The hackathon dispatch flagged that an open-weight model might give lazy or degenerate non-answers that would masquerade as language-conditioned MISSes. We anticipated this by adding an effort-quality classifier (engaged | lazy | refusal | empty | truncated) so non-answers can be separated from real misses. On this run, **every cell across all three models classified as engaged** (Table 4). None of the observed MISSes were lazy, refusal, or empty. The open-weight model’s responses were systematically shorter than the frontier models’ (Llama median ~2350 chars vs Claude median ~4270 chars) but still substantive. This is itself a model-capability finding worth surfacing: the open-weight model is not *lazy* on this prompt and at `max_tokens = 4000`, but it is *terser*, and the one engaged MISS we observed (Llama 3.3 70B on C5 Loopscale / CS) was a generic-finding miss. The model named “oracle manipulation” without identifying the specific

TWAP-floor clamp mechanism. We report this as a real language-conditioned MISS, not a capability artifact, because the same model caught the same mechanism on the same twin in EN, ES, and PT. The variable that flipped was the code-switched comment register.

(f) Open-weight-tier coverage is partial. Seven of the planned 32 llama-3.3-70b-versatile cells (five of them on the largest twin, C4 Mango) could not complete in the build window because Groq’s free-tier daily-token-cap was exhausted mid-run (97,776 of 100,000 tokens used; Groq’s error said retry in 30m41s for the 24h rolling window). The harness is resumable and a follow-up run after the cap rolls over can complete the matrix; the headline cross-model finding (frontier robust, open-weight shows early CS-language sensitivity on C5 Loopscale) is supported by the 25/32 completed open-weight cells but should be re-checked against the full 32/32 once the missing 7 cells land. The full per-cell coverage map is in Appendix A (a – cell is missing, not zero).

(g) Dual-use. The most direct misuse path is “deliberately hide bugs behind non-English comments so AI auditors miss them.” This concern is mitigated by three structural facts: (1) the Atlas twins used here are synthetic, public, and well-documented, using them as an adversarial example offers no incremental capability to a bad actor over reading the original post-mortems; (2) the auditor’s bug-detection ability depends primarily on its code-comprehension, not on comment language; even on a CATCH cell the response identifies the mechanism from the code structure as much as from any comment hint; (3) the framing of this paper is the *deployment* risk to LatAm users, and the appropriate mitigation is on the *model-evaluation* side (add multilingual suites), not on changing what comments LatAm engineers should be allowed to write in their own language.

7. References

1. Yoo, J.; Yang, S.; Lee, J. “Code-Switching Red-Teaming: LLM Evaluation for Safety and Multilingual Understanding.” ACL 2025. (Title and authors per the organizers’ Resources tab; conference proceedings link verified.)↵
2. Banerjee, S. et al. “Bridging the Multilingual Safety Divide.” 2026 working paper, per the Apart Global South Hackathon resources page.↵
3. CaliperForge. “Exploit→Invariant Atlas v0.1.” Internal repository at experiments/invariant-atlas/, 2026; clean/planted same-source twin reconstructions for Cashio (Mar 2022), Mango Markets (Oct 2022), Loopscale (Apr 2025), zkLend (Feb 2025),

and others. Bug-class reconstructions cite the corresponding incident post-mortems (rekt, Helius “Solana Hacks: A Complete History,” BlockSec, Halborn, The Block, Cryptopolitan).↵

4. Yoo, J.; Yang, S.; Lee, J. “Code-Switching Red-Teaming: LLM Evaluation for Safety and Multilingual Understanding.” ACL 2025. (Title and authors per the organizers’ Resources tab; conference proceedings link verified.)↵
5. Banerjee, S. et al. “Bridging the Multilingual Safety Divide.” 2026 working paper, per the Apart Global South Hackathon resources page.↵
6. CaliperForge. “Exploit→Invariant Atlas v0.1.” Internal repository at `experiments/invariant-atlas/`, 2026; clean/planted same-source twin reconstructions for Cashio (Mar 2022), Mango Markets (Oct 2022), Loopscale (Apr 2025), zkLend (Feb 2025), and others. Bug-class reconstructions cite the corresponding incident post-mortems (rekt, Helius “Solana Hacks: A Complete History,” BlockSec, Halborn, The Block, Cryptopolitan).↵
7. CaliperForge. “Exploit→Invariant Atlas v0.1.” Internal repository at `experiments/invariant-atlas/`, 2026; clean/planted same-source twin reconstructions for Cashio (Mar 2022), Mango Markets (Oct 2022), Loopscale (Apr 2025), zkLend (Feb 2025), and others. Bug-class reconstructions cite the corresponding incident post-mortems (rekt, Helius “Solana Hacks: A Complete History,” BlockSec, Halborn, The Block, Cryptopolitan).↵
8. Cashio fake-mint exploit, March 2022, ~\$50M; reconstructed at `cases/03-solana-cashio/` per Helius and rekt write-ups.↵
9. Loopscale PT-collateral oracle manipulation, April 2025; reconstructed at `cases/05-solana-loopscale/` per The Block / Cryptopolitan / Loopscale post-mortem notes.↵
10. Mango Markets oracle-pump exploit, October 2022; reconstructed at `cases/04-solana-mango/` per Helius and the Mango DAO incident post-mortem.↵
11. zkLend share-inflation exploit, February 2025; reconstructed at `cases/01-cairo-zklend/` per BlockSec / Halborn / zkLend post-mortems.↵
12. Yoo, J.; Yang, S.; Lee, J. “Code-Switching Red-Teaming: LLM Evaluation for Safety and Multilingual Understanding.” ACL 2025. (Title and authors per the organizers’ Resources tab; conference proceedings link verified.)↵
13. Banerjee, S. et al. “Bridging the Multilingual Safety Divide.” 2026 working paper, per the Apart Global South Hackathon resources page.↵