Passing Is Not Shipping

ReDoS Risk Tracks Execution and Not Authorship:
Eleven Language Models and Six Populations of Regular Expressions

Plicara Labs
info@plicara.ai

August 2026

Abstract

We evaluate eleven contemporary language models on natural-language-to-regex generation over 450 tasks with three samples each, scoring every answer on three axes: whether it satisfies the task's tests, whether it denotes the same language as a human reference, and whether it is vulnerable to regular-expression denial of service (ReDoS).

Our principal result concerns what determines ReDoS risk. We apply one screen to six populations totalling more than a million patterns: this corpus's human reference answers, a second natural-language-to-regex gold set, a grammar-generated control, a reusable-pattern library, Stack Overflow answers, and regular expressions extracted from shipped packages. Controlling for task mix by restricting every population, models included, to anchored patterns, the resulting ordering is monotone in whether a pattern was ever executed. Patterns *written to be read* are ReDoS-prone: reusable library entries 20.1%, forum answers 17.3%, benchmark reference answers 13.4%. Patterns *executed* in shipped packages sit at 8.9%, and the models sit with them at 9.8% (two-proportion z, $p = 0.20$). **Vulnerability tracks whether a pattern was ever run, and not whether a human or a model wrote it.** We calibrate the screen against an independent detector and a dynamic oracle, and find its differential miss rates real but far too small to produce the ordering.

Second, the correctness-to-security penalty in this domain is small. Just 7.4% of functionally correct patterns are ReDoS-vulnerable, so requiring correctness and safety together costs 2.9 percentage points. Joint correctness-and-security benchmarks in general code generation report far larger penalties: BaxBench finds roughly half of correct backends exploitable, and SecureAgentBench reports 15.2% correct-and-secure for its best agent. The regex domain does *not* reproduce that pattern. Re-running the two reference-independent criteria on a second corpus [Ye et al., 2020] with the same models and configuration shows the finding holds in kind and not in size, 16.5% against 7.4% on tasks twenty points *easier*, so the penalty is a property of the corpus and not of the models. The eleven-model field is compressed into a 6.7-point band that a hundredfold range in inference price does not order.

Third, we quantify what a reference-dependent metric costs. Metrics that compare generated output against a human answer key inherit that key's error rate, a dependency documented for image and text benchmarks [Northcutt et al., 2021] but not, to our knowledge, quantified for this one. In a random sample of disagreements where a model passed every test yet was scored as semantically different, the model was clearly at fault in roughly 15% of cases. The rest split between demonstrably incorrect reference patterns (including a reference character class containing a literal `|`) and prompts that never specified the property in dispute. A composite conjoining correctness, safety and equivalence therefore draws 87% of its headline gap from its least reliable term, and it further awards free credit wherever equivalence is undecidable, which rewards putting an answer beyond checking. We also show that independent binomial intervals, the default in this literature, are the wrong inferential tool when all systems answer identical

items, and that a paired bootstrap resolves nine times as many pairwise comparisons on the same data.

We release all raw generations, scoring code and adjudications, and recommend that evaluations separate reference-dependent from reference-independent metrics, assign them different evidentiary weight, and treat any effect size as corpus-local until replicated.

Artifacts. All model outputs, scoring code, per-case adjudications and analysis scripts are available at `https://github.com/plicara/regexeval-2026` [Plicara Labs, 2026b]. Every number in this paper recomputes offline from committed data with no API access.

1 Introduction

Regular expressions make an unusually good probe for language-model evaluation. The task is compact, the output runs, correctness is partly decidable, and a syntactically valid, test-passing answer can still be a security defect.

The standard protocol for this task scores a candidate against a reference by DFA equivalence [Locascio et al., 2016, Kushman and Barzilay, 2013], sometimes alongside test-based correctness. Neither criterion notices catastrophic backtracking. A pattern can satisfy every provided example, denote exactly the reference language, and still admit inputs on which matching takes time exponential in the input. That is the ReDoS vulnerability class [Davis et al., 2018], a denial-of-service vector in deployed systems and common enough in human-written code to have prompted ecosystem-scale study.

Re(gEx|DoS)Eval closed that gap [Siddiq et al., 2024a]. It introduces the corpus we use, defines the four metrics we use (PASS@k, VULNERABLE@k, DFA-EQ@k and exact match), scores correctness and security jointly on them, and evaluates T5, Phi-1.5 and GPT-3.5-Turbo. The instrument in this paper is theirs. We contribute what happens when that instrument is turned on the current model population, extended to populations of regular expressions that no model produced, and calibrated against detectors and oracles outside itself.

1.1 Contributions

1. **A ReDoS baseline across six populations, and the variable that orders it** (Section 4.5). We apply the identical safety screen to the corpus's own reference patterns, a second NL-to-regex gold set, a grammar-generated control, a reusable-pattern library, Stack Overflow, and half a million regular expressions from shipped packages. Restricting every population, models included, to anchored patterns leaves an ordering that is monotone in whether a pattern was ever executed and flat in who wrote it. Screening costs no inference, so this result generalises where the model-side ones cannot, and Section 4.5.2 puts it against an independent detector and a dynamic timing oracle.

2. **A measurement of the correctness-to-security penalty in a domain that does not follow the joint-benchmark pattern** (Sections 4.3 and 4.6). 7.4% of functionally correct patterns are ReDoS-vulnerable here, against roughly half of functionally correct backends in BaxBench and 15.2% correct-and-secure for SecureAgentBench's best agent. We report that difference and test the deflationary explanations for it.

3. **A second-corpus replication** (Section 4.7). We re-run the two reference-independent criteria on StructuredRegex [Ye et al., 2020] with the same models and configuration. The finding replicates in kind and not in size: 16.5% against 7.4%, on easier tasks, with the model

ordering not preserved. This is the direct test of Section 4.6's domain-dependence claim, which otherwise rests on comparison across languages and vulnerability classes.

4. **An updated model population.** The original Re(gEx|DoS)Eval study evaluates T5, Phi-1.5 and GPT-3.5-Turbo [Siddiq et al., 2024a]. We evaluate eleven models released in the intervening period under pinned serving providers (Section 3.3), and find the population compressed into a 6.7-point band that a hundredfold price range does not order.

5. **A quantified audit of reference-standard error** (Section 5.1). We sample disagreements and adjudicate them, finding that the majority of apparent semantic errors are not model errors. Test-set label error is pervasive and can reverse benchmark orderings [Northcutt et al., 2021]; we measure it for a corpus whose labels are regular expressions, where a label error is a formal object that can be exhibited and argued with rather than a disputed annotation. The corpus's own metrics are reported without an audit of the reference set they compare against [Siddiq et al., 2024a], which is the standard practice we are questioning rather than a defect specific to them.

6. **A decomposition of the joint metric** (Section 4.3). Conjoining correctness, safety and equivalence is the natural way to report them together, and it is what the joint-benchmark literature reports (Section 2.1). On this corpus the composite draws 87% of its gap from the equivalence term audited in Section 5.1, and awards free credit wherever equivalence is undecidable (Section 5.2). We recommend against the construction and give the decomposition that replaces it.

7. **Paired inference for benchmarks where every system answers the same items** (Section 5.3). Independent binomial intervals, the default in this literature, discard the pairing. A paired bootstrap resolves 9 of fifty-five pairwise comparisons against one on the same data.

8. **Full artifact release** including raw generations, which permits any of our scoring decisions to be overturned without re-running inference.

2 Background and Related Work

NL-to-regex generation. The task was established by KB13 [Kushman and Barzilay, 2013] and extended by NL-RX [Locascio et al., 2016], both of which adopt DFA equivalence against a reference as the correctness criterion. That is the right choice in principle. `[0-9]+` and `[0-9][0-9]*` denote the same language and differ as strings, so text comparison would mark one of them wrong. But the criterion inherits any defect in the reference set, which is the dependency this paper quantifies.

Re(gEx|DoS)Eval. Re(gEx|DoS)Eval supplies both the corpus and the metric suite used here [Siddiq et al., 2024a]. Its authors collect 762 tasks from real user posts, each with a description, positive and negative test strings and a human reference pattern, and score generations on PASS@k, VULNERABLE@k, DFA-EQ@k and exact match, reporting which model produces regexes that are correct *and* secure. Their evaluation covers T5, Phi-1.5 and GPT-3.5-Turbo. Everything this paper measures is measured with their instrument. Our departures are three: we run it on eleven models released since, we decompose the joint metric instead of reporting it whole (Section 4.3), and we turn the safety screen on their reference patterns (Section 4.5), which they do not do. The companion study [Siddiq et al., 2024b] examines ReDoS in LLM-generated patterns alongside developer-forum discussion and reports a skew toward polynomial over exponential vulnerability; we compare against that finding in Section 4.5.

ReDoS. Catastrophic backtracking arises when a backtracking matcher can decompose a prefix in exponentially many ways, canonically via a quantifier applied to a quantified group, as in `(a+)+`. Detecting it statically is a developed area: exponential blow-up admits an analysis by search over the pattern's derivation tree [Kirrage et al., 2013], detection extends to programs that construct patterns dynamically [Wüstholz et al., 2017], and the detector and mitigation literature has been surveyed [Bhuiyan et al., 2025]. The scoring engine we depend on [Plicara Labs, 2026a] sits in that lineage and combines a structural screen with witness-string execution. On prevalence, ReDoS is documented at ecosystem scale [Davis et al., 2018] and regex portability and re-use have been studied across languages [Davis et al., 2019].

Security of generated code. The observation that models emit working but unsafe code predates the joint benchmarks below. A sweep of 1,689 generated programs across 89 security-relevant scenarios found roughly 40% vulnerable [Pearce et al., 2022], and a controlled user study showed that participants with an AI assistant wrote less secure code while believing the opposite [Perry et al., 2023]. Both motivate scoring security separately from correctness rather than assuming the first implies the second.

2.1 Joint correctness-and-security benchmarks

Evaluating functional correctness and security together is an active area, and this paper is not its first instance. CWEval scores 119 security-critical tasks across 31 CWEs and five languages with outcome-driven oracles for both properties, and reports a substantial population of functionally correct but insecure generations [Peng et al., 2025]. BaxBench pairs 392 backend tasks with expert exploits and finds that roughly half of functionally correct solutions are exploitable [Vero et al., 2025]. SecureAgentBench evaluates coding *agents* on 105 repository-level tasks, reporting 15.2% correct-and-secure for the best agent and 9.2% on average [Chen et al., 2025]. DualGauge automates the joint pipeline over 154 tasks and 10 models, observing secure-pass@1 below 12% while functional pass@1 exceeds 50% [Patir et al., 2025].

The joint framing is therefore established both across domains and on this one [Siddiq et al., 2024a]. What differs here is the shape of the instrument. The artifact is a single expression instead of a program. The vulnerability class is ReDoS instead of a CWE taxonomy. Correctness is partly decidable, not established by tests alone, and a human reference exists for every task, which the general-code benchmarks do not have: they execute tests and exploits rather than compare against a gold artifact. That difference cuts both ways. It makes Section 4.5 possible, and it is the sole reason Section 5.1 is necessary. The reference-free benchmarks are immune to the failure mode we spend half this paper characterising.

Our numbers turned out not to match theirs. Section 4.6 works through the difference.

What this paper takes from that work. The apparatus is inherited. The corpus, the task set and the metric definitions (VULNERABLE@k, DFA-EQ@k, exact match) come from Re(gEx|DoS)Eval [Siddiq et al., 2024a]; PASS@k is the standard unbiased estimator [Chen et al., 2021]. DFA-equivalence checking is the field's standard correctness criterion [Locascio et al., 2016], and the ReDoS screen sits in the static-analysis lineage above [Kirrage et al., 2013, Wüstholz et al., 2017, Bhuiyan et al., 2025], implemented in a scoring engine that is prior work by the present authors, released separately and used here as a pinned dependency [Plicara Labs, 2026a]. What this paper builds on top of that inheritance is the six-population extension, the calibration of the screen, the audit of the reference set, and the paired inferential treatment; the USABLE composite of Section 3.4 is a conjunction of

the inherited metrics, and Section 4.3 is the argument that it should be decomposed rather than reported.

Sampling-based metrics. We use the unbiased PASS@k estimator [Chen et al., 2021], with the degeneracy noted in Section 3.4.

Significance testing. Comparing systems that answer identical items calls for a paired procedure. The paired bootstrap we use comes from the empirical-methods literature [Berg-Kirkpatrick et al., 2012], following the bootstrap-resampling protocol for machine translation [Koehn, 2004], and its use over unpaired intervals is standard advice [Dror et al., 2018]; the discrete analogue for paired binary outcomes is McNemar's test [McNemar, 1947]. Section 5.3 reports what the choice costs on this corpus rather than proposing anything new.

Contamination. This corpus was published in 2024 and its tasks come from public forum posts, so it is plausibly inside the training data of every model we evaluate. Contamination must be measured per benchmark rather than assumed away [Sainz et al., 2023]. We cannot measure it here and say so in Section 6.

Serving-layer variance. Hosted inference routers may serve the same model slug from multiple providers at differing numerical precision. A survey of AI-safety codebases using one such router found that 31 of 32 did not pin the serving provider [LessWrong, 2025], making their results a measurement of the router's routing policy in addition to the model. Section 3.3 describes the discipline we adopt in response.

3 Method

3.1 Task and prompt

Each task supplies a natural-language description. The model returns one regular expression. The prompt is fixed across all models and all tasks:

> **system:** You translate a natural-language description into a single Python `re`-compatible regular expression. Reply with ONLY the pattern in one fenced code block, no explanation.
>
> **user:** *{task description, verbatim}*
>
> Reply with only the regular expression pattern in a single fenced code block.

Prompt optimisation would raise all scores and destroy comparability with prior work on the same corpus, so the prompt is held fixed; Section 6 notes the corresponding threat.

Extraction. We take the final fenced code block, else the whole trimmed reply, then strip host-language quoting (`r'...'`, `'...'`, `"..."`, backticks, `/.../flags`), repeatedly up to three nested layers so a backtick wrapped around a raw string arrives bare. Scored literally, `r'\d+$'` is a pattern matching the letter `r`, and would fail for reasons unrelated to regex competence. This affected 7–18 of 1,350 responses per model, under 1.4% throughout, and changes no conclusion. We release both the normalised and the unnormalised scores.

3.2 Corpus

We use Re(gEx|DoS)Eval [Siddiq et al., 2024a], 762 tasks drawn from real user requests, each with a description, positive and negative test strings, and a human-written reference pattern. We evaluate 450 tasks, selected at uniform stride across the corpus rather than as a prefix (the corpus is difficulty-ordered at its head). The subset size was budget-determined.

Match semantics are set per corpus. Under search semantics 761 of the 762 Re(gEx|DoS)Eval references satisfy their own tests, against 94% under full-match. Choose wrong and 46 gold patterns are scored as failures. We verified the loaded configuration by scoring every reference against its own tests before the run (Section 5.1).

The single exception is not a mismatch. `regexeval/1660`'s reference, `^((\.)?([a-zA-Z0-9_-]?)(\.)?([a-zA-Z0-9_-]?)(\.)?)+$`, does not accept a string it should reject: it fails to *finish* on one, exceeding the scorer's one-second match timeout, which is counted as a false positive. The safety screen independently flags the same pattern as exponential. The one reference in the corpus that fails its own tests fails because it is ReDoS-vulnerable.

3.3 Models and serving discipline

Eleven models spanning frontier and open-weights systems across a 98× price range. Every request pins a single named provider with fallback disabled, so a request is either served by the named endpoint or fails visibly. Substitution failures did occur and are reported as failures rather than silently re-routed (Section A).

For open-weights models the pin includes quantisation where the router discloses it: GLM-5.2 and DeepSeek-V4-Flash are served at 4-bit by some providers and 8-bit by others, and we pin 8-bit. All eleven models came back served entirely by the endpoint they were pinned to.

3.4 Metrics

Let $T = (d, P, N, r)$ be a task with description d, positive examples P, negative examples N and reference pattern r. Let p be a candidate.

Correctness.

$$\text{correct}(p, T) \;=\; [\forall x \in P :\; \text{match}(p, x)] \;\wedge\; [\forall y \in N :\; \neg\, \text{match}(p, y)]$$

evaluated with CPython's `re` under the corpus's declared semantics. This is *reference-independent*.

Semantic equivalence. Both p and r are compiled to finite automata and compared via `dk.brics.automaton` [Møller, 2021], yielding a verdict in $\{\text{EQ}, \text{DIFF}, \text{UNDEC}, \text{UNSUP}\}$. When the verdict is DIFF the engine returns a *witness*: a shortest string in the symmetric difference, which renders the judgment falsifiable by inspection.

Equivalence is decidable only on the regular subset. Backreferences make the language non-regular, and equivalence then has no algorithmic answer at all. Those cases return UNDEC. A pattern the engine declines for any other reason (a lookahead containing an anchor, an inline flag group, an automaton over a state budget) returns UNSUP. Both are reported together in this paper's *undec.* column, because USABLE gives them the same free credit and the accounting has to include both, but they are different claims: one is a theorem about backreferences, the other is this engine's

reach. Between 78 and 133 of each model's scored tasks land in the pair; Section 5.2 gives the split, and it falls almost entirely on one side. We therefore report two figures:

$$\textsc{dfa-eq} = \Pr[\textsc{eq}], \qquad \textsc{dfa-eq}_{\text{dec}} = \Pr[\textsc{eq} \mid \text{verdict} \neq \textsc{undec}].$$

The first counts undecidable comparisons as failures. It is a lower bound and cannot flatter. The second isolates the model from the engine's reach. Report only the second and you inflate. Report only the first and you charge the model for a theorem. This metric is *reference-dependent*, which Section 5.1 shows to be decisive.

ReDoS safety. $\text{vuln}(p) = [\text{screen}(p) \neq \textsc{safe}]$, where screen performs structural analysis for known-vulnerable shapes followed by an empirical pass against attack strings under timeout. The structural pass covers three of the five families catalogued in prior work on this corpus [Siddiq et al., 2024b]. So VULNERABLE is a *lower bound*, and SAFE denotes a screening outcome, not a proof. This metric is *reference-independent*. Section 4.5.2 measures how loose the bound is, separately for each population we compare, because a lower bound that is looser in one population than another would produce an ordering by itself.

Where the screen reports a degree, exponential against polynomial, the structural pass infers it from the matched shape and the empirical pass infers it from which probe length first exceeded the timeout. The second is a threshold rather than a measurement of growth order, and we discount the degree split accordingly (Section 4.5.2).

Composite.

$$\text{usable}(p, T) = \text{correct}(p, T) \ \wedge \ \neg\, \text{vuln}(p) \ \wedge \ \big[\text{verdict}(p, r) \neq \textsc{diff}\big]$$

The third conjunct excludes only *proven* difference. Undecidable and unsupported verdicts do not disqualify a pattern. USABLE is reference-dependent through that conjunct, and inherits the error characterised in Section 5.1.

Sampling. With n samples per task of which c satisfy a predicate, we use the unbiased estimator [Chen et al., 2021]:

$$\widehat{\text{metric}@k} \ = \ \mathbb{E}_T \left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}} \right].$$

We draw $n = 3$ and report $k = 3$. At $n = k$ this estimator degenerates to $\mathbf{1}[c \geq 1]$ and buys no variance reduction over the naive any-of-n statistic. We report it as @3 for comparability, not because the estimator is doing any work at this setting. $n > k$ would be required for the estimator's variance properties to apply, and is left to future work with a larger budget.

The estimator is defined for $n \geq k$, and a handful of tasks returned fewer samples than that after a refusal or a spending limit. We exclude those tasks from the affected model's @k estimate rather than score them on what arrived, and report the count in Table 3. Section B records why this is not a cosmetic choice.

3.5 Sampling configuration

Reasoning disabled. Four of the eleven models emit no hidden reasoning tokens. The other seven do. Evaluating them together with reasoning enabled would confound model identity with inference-time compute budget. All requests therefore disable reasoning, and reasoning-token counts are recorded per response to verify compliance (observed: zero throughout).

Temperature unset. We set `require_parameters=true`, which restricts routing to providers that honour submitted sampling parameters rather than silently discarding them. Six of the eleven models reject `temperature` outright. Together, the two settings make the request unroutable. We therefore omit `temperature` entirely and each model samples at its provider default. That keeps the non-silent-discard guarantee and costs us controlled sampling. We think it is the right trade and note it as a limitation.

Since $k > 1$ is only informative under output diversity, we verified diversity directly: repeated queries on a fixed task yielded 3, 3, 3 and 2 distinct patterns for `claude-opus-5`, `gpt-5.6-sol`, `kimi-k3` and `glm-5.2` respectively. The analysis pipeline also warns if $> 90\%$ of a model's tasks return identical samples.

3.6 Controls

Three synthetic responses go through the identical scoring path on every run. The task's own reference must come back correct and usable. The pattern `z{5}` must fail every correctness check. `(a+)+b` must be flagged vulnerable. If any control misbehaves we throw the run away instead of reporting it.

The failure this catches is a scorer that quietly returns nulls, which looks exactly like every model performing badly. All controls behaved on all eleven models.

4 Results

4.1 Summary of findings

Let us state the four results first and spend the rest of the section establishing them.

1. **ReDoS prevalence is ordered by execution, not by authorship.** Across six populations restricted to a common pattern shape, regular expressions written to be read screen vulnerable at 13–20%, while those executed in shipped packages sit at 8.9% and the models sit with them at 9.8% (Section 4.5). The screen survives calibration against an independent detector and a dynamic oracle (Section 4.5.2).

2. **The correctness-to-security penalty here is small, and it is a property of the corpus.** 7.4% of functionally correct patterns are ReDoS-vulnerable, costing 2.9 points to require both (Section 4.3). Comparable joint benchmarks in general code generation report penalties an order larger (Section 4.6), and the same models on a second regex corpus give 16.5% (Section 4.7).

3. **The field is compressed and price does not order it.** Eleven models span 6.7 points of USABLE@3 across a 98× range in inference cost, and the cheapest and dearest models differ by 1.1 points (Section 4.8).

4. **Most of the apparent correct-to-shippable gap is contributed by the equivalence term**, which Section 5.1 then shows to be the least reliable of the three (Section 4.3).

4.2 Main results

Table 1 gives the three axes separately and their conjunction. The conjunction is the quantity the joint-benchmark literature reports, and on this corpus it falls from PASS@3 of 38.0–46.5% to

Model	n	PASS@3	USABLE@3	95% CI	VULNERABLE@3	DFA-EQ@3	DFA-EQ$_{\text{dec}}$	EXACT@3	undec.
`kimi-k3`	441	46.5	23.8	[20.0, 27.9]	12.9	14.1	17.1	5.0	78
`qwen3.6-max-preview`	450	42.4	21.6	[17.8, 25.6]	9.1	13.8	17.4	3.8	94
`gpt-5.6-sol`	449	42.1	20.9	[17.1, 24.7]	10.0	9.4	13.3	1.8	133
`claude-opus-5`	432	46.1	20.8	[16.9, 24.8]	13.2	11.6	15.2	2.8	104
`deepseek-v4-flash-0731`	450	38.0	19.8	[16.2, 23.6]	12.0	11.3	14.3	3.3	93
`qwen3.6-plus`	450	39.8	19.8	[16.2, 23.6]	9.8	11.6	14.8	3.8	99
`glm-5.2`	450	42.4	18.7	[15.1, 22.4]	14.2	10.4	12.9	3.8	86
`gpt-5.6-terra`	450	42.2	18.7	[15.1, 22.4]	12.0	10.2	13.1	2.0	100
`gpt-5.6-luna`	449	39.2	18.5	[14.9, 22.0]	11.6	9.1	12.2	1.8	112
`claude-sonnet-5`	450	40.7	18.0	[14.4, 21.6]	10.9	10.4	13.3	3.3	97
`gemini-3.1-flash-lite`	450	38.7	17.1	[13.8, 20.7]	12.0	9.3	11.8	3.1	95

Table 1: Primary results, $n = k = 3$. All figures percentages except n (tasks entering the @3 estimate) and *undec.* (tasks whose equivalence is undecidable). Tasks that returned fewer than three samples are excluded from the estimate rather than scored on what arrived (Section 3.4), which is why n varies; failed requests are itemised in Section A. The interval is a task-level bootstrap of each model's own score. It is *not* the paired procedure of Section 5.3 and carries none of its benefit: pairing is a property of a comparison, not of a score, so a marginal interval here lands within a twentieth of a point of the binomial one it might be mistaken for. **Do not read overlap between these intervals as a difference that does not resolve.** That inference is the error Section 5.3 exists to correct, and 9 pairs separate at 95% under the paired test while every interval in this column overlaps. Ordering is by USABLE@3 and should be read as a banding, not a ranking (Section 5.4).

USABLE@3 of 17.1–23.8%: approximately half of all test-passing generations fail at least one of the remaining two criteria, and the ratio is stable across the entire price range.

An aggregate of that shape is only as good as its weakest term, so we do not report it on its own. Section 4.3 attributes the drop term by term, and that attribution is what the rest of the paper builds on.

4.3 Decomposing the gap

USABLE conjoins three conditions. Table 2 attributes the drop from PASS@3 to USABLE@3 between the safety term and the equivalence term, and reports alongside it the rate of ReDoS vulnerability *conditional on functional correctness.*

Safety costs 2.9 percentage points on average. Equivalence costs 18.9. So the equivalence term supplies **87%** of the gap that made the composite look interesting.

That split is order-dependent, and the order we chose is the one that favours the other term. Subtracting safety first credits it with every unsafe-correct pattern, including those that would also have failed equivalence, while equivalence keeps only the share no other term had already claimed. Reversing the order can only move attribution toward equivalence. 87% is therefore a *lower bound* on the equivalence term's contribution under any order-symmetric attribution. The obvious objection to a sequential decomposition is that it is order-dependent; here the dependence runs against us.

That would be unremarkable if the equivalence term were sound. It is not. Section 5.1 shows, from an independent audit, that roughly 85% of what it counts against a model is reference-set error or an underspecified prompt. Our composite was mostly reporting its own weakest component back to us.

Model	PASS@3	−safety	C&S@3	−equiv.	USABLE@3	vuln.|correct
`kimi-k3`	46.5	2.5	44.0	20.2	23.8	5.6
`qwen3.6-max-preview`	42.4	2.7	39.8	18.2	21.6	7.0
`gpt-5.6-sol`	42.1	1.6	40.5	19.6	20.9	5.6
`claude-opus-5`	46.1	4.9	41.2	20.4	20.8	10.0
`deepseek-v4-flash-0731`	38.0	1.8	36.2	16.4	19.8	5.7
`qwen3.6-plus`	39.8	3.3	36.4	16.7	19.8	8.1
`glm-5.2`	42.4	4.0	38.4	19.8	18.7	9.3
`gpt-5.6-terra`	42.2	2.9	39.3	20.7	18.7	7.3
`gpt-5.6-luna`	39.2	2.9	36.3	17.8	18.5	8.1
`claude-sonnet-5`	40.7	2.7	38.0	20.0	18.0	6.0
`gemini-3.1-flash-lite`	38.7	2.9	35.8	18.7	17.1	8.3
mean	—	2.9	—	18.9	—	7.4
pooled	—	—	—	—	—	**7.4**

Table 2: Decomposition of the PASS→USABLE gap. "C&S" is correct-and-secure: correctness conjoined with safety, omitting the equivalence term. The final column is the share of functionally correct patterns that are ReDoS-vulnerable, and is the one column here computed *per sample* rather than @3: it is a rate over generations, and the @3 form would answer a different question. Denominators are Table 3's restricted ones (tasks with fewer than three samples are excluded), which is why `kimi-k3` reads 46.5 here.

Restricting attention to the two reference-independent criteria, the finding is far more modest and far better supported: **7.4% of functionally correct regular expressions are ReDoS-vulnerable** (range 5.6–10.0%).

The denominator matters, so: of the 5,269 generations that satisfied every test, 390 were simultaneously screened as unsafe. That is 7.4%, a rate over *samples* rather than over tasks, which makes it an @1 quantity and the one directly comparable to the single-sample figures in Table 9. At the task level the corresponding count is 144 of 2,051 tasks on which some sample passed. About one in fourteen either way, not one in two. Both counts and both denominators recompute from the released per-sample records, which is the discipline Section C argues for.

EXACT@3, string identity with the reference, runs from 1.8% to 5.0%. String-comparison scoring would therefore misrank essentially every system, which is what justifies the automata-theoretic treatment.

4.4 Sample budget and the fragility of ordering

Because $n = k = 3$ renders the PASS@k estimator degenerate (Section 3.4), we also report @1, which is the per-sample success rate and the quantity most directly comparable to single-sample protocols. Table 3 gives both.

The comparison is instructive beyond bookkeeping. At @3, `kimi-k3` attains the highest USABLE at 23.8%. At @1 the leader is `claude-opus-5` on 17.3%, and `kimi-k3` drops to joint third. *The induced ordering depends on the sample budget.* Systems differ not only in per-attempt quality but in how much they benefit from additional attempts, and a leaderboard that fixes k silently fixes a weighting between those two properties. This is a further reason to prefer banded reporting (Section 5.4), and a reason to state k prominently whenever an @k metric is ranked.

Model	PASS@1	PASS@3	USABLE@1	USABLE@3	VULNERABLE@1	n<3
`kimi-k3`	35.6	46.5	16.0	23.8	9.2	6
`qwen3.6-max-preview`	37.0	42.4	16.5	21.6	7.4	0
`gpt-5.6-sol`	37.3	42.1	16.0	20.9	8.4	1
`claude-opus-5`	41.4	46.1	17.3	20.8	10.7	12
`deepseek-v4-flash-0731`	30.0	38.0	14.4	19.8	7.7	0
`qwen3.6-plus`	35.5	39.8	15.9	19.8	8.0	0
`glm-5.2`	34.1	42.4	13.5	18.7	9.6	0
`gpt-5.6-terra`	36.6	42.2	14.8	18.7	8.9	0
`gpt-5.6-luna`	33.8	39.2	15.2	18.5	8.7	1
`claude-sonnet-5`	36.8	40.7	15.3	18.0	9.3	0
`gemini-3.1-flash-lite`	33.2	38.7	13.9	17.1	9.5	0

Table 3: Per-sample (@1) against any-of-three (@3) estimates, computed from per-sample success counts. The final column gives the number of tasks with fewer than three usable samples, which are excluded from that model's @3 estimate. The @3 columns agree with Table 1 to the digit for all eleven models; they are computed by different routes over the same exclusion, and Section B records the estimator defect this check caught.

4.5 ReDoS and the human baseline

Table 4 applies the identical safety screen to the corpus's 450 human-written reference patterns. For comparability with the human set (one pattern per task) we score one sample per task per model rather than the any-of-three statistic of Table 1.

Every model screens safer than the reference set, and 6 of the eleven do so distinguishably once the eleven tests are corrected for multiplicity. That second clause is the one that needs the work, because the obvious way to get it is wrong twice over. Reading the ordering off eleven independent binomial intervals over a pooled 4,941 patterns commits both of the errors this paper criticises elsewhere. The 4,941 are eleven models answering the same 450 tasks, so task difficulty is a shared random effect and a pooled binomial interval is too narrow (Section 5.3), and each model is compared against the reference set *on the same tasks*, so the comparison is paired and an unpaired interval discards the pairing: with $n \approx 450$ a side and a three-point difference, nothing would resolve.

McNemar's test is the paired procedure for this design, and we use the exact binomial form because the discordant counts are small (47 to 65 per model). Everything the model and the reference have in common cancels. Against `qwen3.6-max-preview`, for instance, the reference set is vulnerable where the model is safe on 39 tasks and safe where the model is vulnerable on 11 ($p = 0.0001$). Against `kimi-k3` the split is 39 to 26 ($p = 0.14$), which does not resolve. The three models that do not separate (`kimi-k3`, `claude-opus-5` and `glm-5.2`) are the three with the highest vulnerability rates, which is what one would expect and is not what an eleven-way ordering of point estimates would have told us.

Eleven tests need a multiplicity correction, which we owe the reader in a paper arguing that people reach for the wrong inferential tool. Holm–Bonferroni at $\alpha = 0.05$ stops at the seventh ordered p-value and resolves 6 of eleven; Benjamini–Hochberg, controlling the false discovery rate instead, resolves 8. We quote the Holm figure in the abstract and the conclusion because it is the conservative one; readers who prefer FDR should read 8.

So: models are safer than this corpus's answer key, on most of the field, under the right test. Read alone, that invites a strong reading: ReDoS vulnerability is endemic to human-written regular expressions, and models merely reproduce it at a lower rate. It does not survive the obvious check:

Source	n	vulnerable (%)	exponential (%)	polynomial (%)	McNemar p
Human reference answers	450	13.6	6.4	7.1	—
`kimi-k3`	447	10.7	5.4	5.4	0.136
`claude-opus-5`	444	10.6	7.4	3.2	0.065
`glm-5.2`	450	10.2	5.8	4.4	0.058
`gemini-3.1-flash-lite`	450	9.8	5.1	4.7	0.033
`gpt-5.6-sol`	450	9.3	5.1	4.2	0.018
`claude-sonnet-5`	450	8.9	6.0	2.9	0.005
`gpt-5.6-terra`	450	8.7	4.9	3.8	0.004
`qwen3.6-plus`	450	8.4	5.1	3.3	0.001
`gpt-5.6-luna`	450	8.0	4.0	4.0	0.001
`deepseek-v4-flash-0731`	450	7.3	4.4	2.9	0.000
`qwen3.6-max-preview`	450	7.3	4.7	2.7	0.000
All models pooled	4941	9.0	5.3	3.8	—

Table 4: ReDoS screening of model outputs against the corpus's own human-written reference patterns, one pattern per task in all cases, that is the first sample and not a mean over the three. The human side has one answer per task, so a mean would compare an average of three attempts against a single attempt; Table 3 reports the three-sample estimators separately, which is why its VULNERABLE@1 column differs from this table's rate. The final column is the exact McNemar test against the reference set on the same tasks; see below for why an independent interval would be the wrong instrument here.

a benchmark answer key is not the same population as working code.

4.5.1 Six populations

Screening a pattern requires no inference, so the human side of this comparison can be extended to any corpus at no cost. We applied the identical screen to five further populations: the gold answers of KB13 [Kushman and Barzilay, 2013]; the targets of NL-RX [Locascio et al., 2016], generated from a grammar rather than written by people and included as a control; and three corpora from the artifact of Davis et al. [Davis et al., 2019], namely regular expressions extracted from shipped packages across eight registries, regular expressions posted to Stack Overflow, and the reusable patterns published on `regexlib.com`. Samples are drawn at a fixed seed.

Production code screens at 4.9%, below every model we evaluated; the endemic reading is wrong.

The raw comparison is confounded by task mix. This corpus is unusually rich in validators (email, ISBN, hostname), the construction that backtracks, whereas most regular expressions in the wild are short fragments with no opportunity to. Restricting every population to anchored `^...$` patterns gives the closest matched populations available, and the ordering that emerges does not follow the axis we expected:

> The dividing line is not human against machine. It is whether the pattern was ever run. Every population written to be *read* (library entries at 20.1%, forum answers at 17.3%, benchmark reference answers at 13.4%) is ReDoS-prone. The one population that has been *executed* under real traffic sits at 8.9%, and the models sit with it at 9.8%.

The model row is restricted the same way. The anchoring rule keys on the pattern and not on its provenance, so the model row reports the models' rate on their own anchored outputs: 9.8%

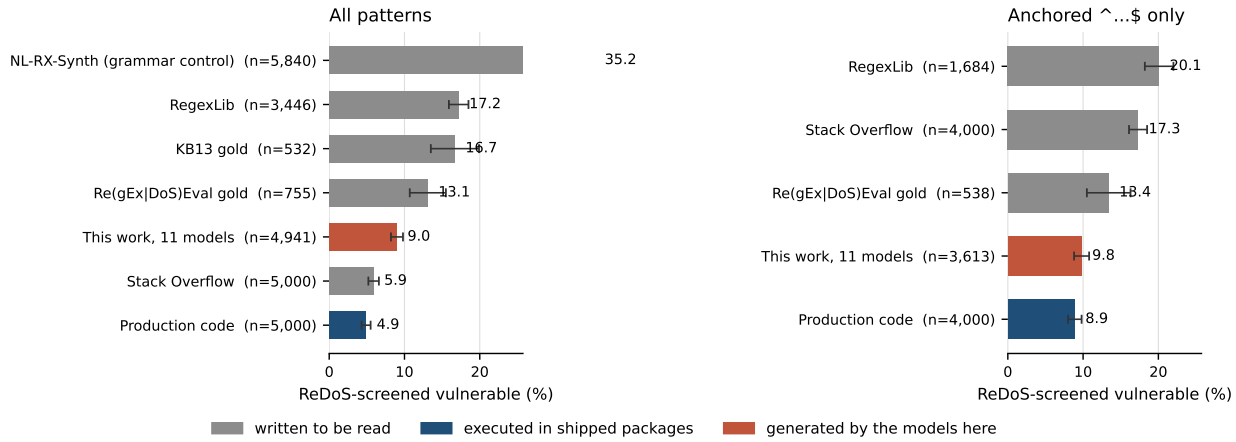

Figure 1: The same screen on every population, before and after controlling for task mix. Bars are 95% binomial intervals. The restriction is the same rule on both sides (keep only anchored `^...$` patterns) and it is applied to the models on the same terms as everyone else. What survives it is not an ordering of humans against machines.

Population	Written to be	n	vulnerable
All patterns			
NL-RX-Synth (grammar-generated control)	—	5,840	35.2% ± 1.2
RegexLib, published for reuse	read	3,446	17.2% ± 1.3
KB13 gold answers	read	532	16.7% ± 3.2
Re(gEx|DoS)Eval gold answers	read	755	13.1% ± 2.4
Stack Overflow posts	read	5,000	5.9% ± 0.7
Production code	run	5,000	4.9% ± 0.6
Anchored `^...$` only, controlling for task mix			
RegexLib, published for reuse	read	1,684	20.1% ± 1.9
Stack Overflow posts	read	4,000	17.3% ± 1.2
Re(gEx|DoS)Eval gold answers	read	538	13.4% ± 2.9
This work, 11 models pooled	—	3,613	**9.8% [7.5, 12.4]**
Production code	**run**	4,000	**8.9% ± 0.9**

Table 5: The identical ReDoS screen applied to six populations, plus this paper's models under the identical restriction. Intervals are 95% binomial for the wild populations, whose patterns are genuinely independent draws. *The model row is not*: its 3,613 patterns are eleven models answering the same 330 anchored tasks, so it carries a bootstrap resampled over tasks, which is 2.4 times wider than the binomial interval the same counts would give. The anchored block is the comparison to read: the unanchored figures are dominated by task mix, which is why Stack Overflow moves from 5.9% to 17.3% between the two blocks.

± 1.0 over 3,613 patterns. Taking the models' *unrestricted* rate from Table 4 instead would set an anchoring-restricted human population against an unrestricted model one, which is the confound the restriction exists to remove. Restricting them costs about eight tenths of a point in the direction of the answer key and away from production code, and 9.8% against production code's 8.9% remains an unresolvable difference.

That comparison needs the right interval, and the obvious one is wrong. The 3,613 anchored model patterns are eleven systems answering the same 330 tasks, so a binomial interval over them treats shared task difficulty as independent draws, which is the error this paper corrects in Section 5.3. Resampled over tasks the interval is $[7.5, 12.4]$ rather than ± 1.0. Against production code that changes nothing, because a wider interval only strengthens "indistinguishable". Against the nearest read population it changes the claim: the anchored gold set's $13.4\% \pm 2.9$ spans $[10.5, 16.3]$, which the models' interval now touches, so *models against gold does not resolve at this matching either*. That is what we already concede for the stricter matchings below, and it should be said uniformly rather than left as the one comparison that separates because it used the looser estimator.

Two stricter matchings are available here that are not available for the wild populations, because we know which *task* each model pattern answers. Restricting to the 330 of 450 tasks whose reference is itself anchored, the models screen at 10.6% against the reference set's 13.9%. Requiring both (an anchored output on an anchored task) gives 11.4% against 13.9%, and at that point the model-versus-reference difference is no longer resolvable either. That is the honest ceiling on how strongly the reference comparison can be stated, and it does not disturb the production comparison, which is a population-level restriction applied identically on both sides.

This is a measurement of the mechanism rather than a conjecture about it. A pattern published for reuse, posted in an answer, or written to key a benchmark is authored once to communicate an idea, and nothing subsequently happens to it. A pattern in a shipped package is executed, profiled, and occasionally repaired; documented ecosystem-scale remediation is exactly that remediation activity at ecosystem scale. Vulnerability tracks exposure to execution, and answer keys have none.

A competing explanation produces the same ordering without any repair. Packages under a lint rule such as `safe-regex` never contained the bad pattern to begin with, so selection at authoring time would look exactly like survival after it. The two are separable in principle (repair leaves a commit and prevention does not) and we cannot separate them here: the artifact we screen carries registry and module counts but no maintenance history, so we can stratify by how widely a pattern is used and not by whether its package is still maintained. Davis et al.'s documented remediation supports the repair mechanism and does not rule out the other. We state the finding as exposure-to-execution because that is what both mechanisms share, and flag the decomposition as open.

It also explains why this corpus's answers are ReDoS-prone without appeal to authorial carelessness. Re(gEx|DoS)Eval was assembled from real user posts, and forum posts screen at 17.3%. That the gold set is safer than its own source population (13.4%) suggests its authors curated, which is to their credit and does not close the gap to executed code.

The control is the clearest case of the mechanism. NL-RX-Synth screens at 35.2%, more than double the next population and seven times production code. That is not the screen misbehaving on an odd construct distribution, and the reasons say so: every one of the 2,057 vulnerable patterns is flagged by the *structural* pass, none by the empirical one, and they divide into 994 with a quantifier wrapping a quantified group, 891 with two quantifiers in sequence over overlapping character sets, and 172 with a quantifier over overlapping alternation. Representative targets are `.*(.*)([a-z]).*`, `(dog.*[0-9].*)+` and `((dog)|(..*[AEIOUaeiou].*)){4,}`.

The generating grammar composes `.*`, `(...)*` and `(...){n,}` freely over overlapping character classes, and those compositions *are* the backtracking shapes. A grammar that samples uniformly from a space of regular expressions has no reason not to, because nothing downstream of it ever runs the output. Far from being an outlier that impugns the screen, NL-RX-Synth is the purest instance of this section's mechanism: it is the population with the least exposure to execution of any we measured, and it is the most vulnerable. Section 4.7 returns to this, because StructuredRegex is grammar-built too.

Section 5.1 reaches the same conclusion from semantics, and this section reaches it from safety without consulting the reference at all: the reference set is the least reliable artifact in the evaluation. The practical implication holds either way. Safety screening must be applied at the point of use, because neither the generator, nor the reference author, nor the benchmark performs it.

4.5.2 Is the screen equally sensitive across these populations?

The ordering above is only a fact about the populations if the screen misses vulnerable patterns at the same rate in all of them. If production patterns are longer and structurally unlike RegexLib's showcase validators, a differential miss rate could generate the entire result. "VULNERABLE is a lower bound" is an adequate caveat for one population and is not one for six, so we measure it.

The screen does not see every dialect. Every population is screened by CPython's `re`, and patterns it will not compile are dropped. That filter is not uniform: it removes 5.0% of the production pool (26,900 of 537,804) against 11.4% of Stack Overflow and 10.2% of RegexLib, and within production it runs from `cpan` at 7.7% down to `pypi` at 0.3%. Section 6 applies exactly this scrutiny to KB13 and NL-RX, whose DSL `re` accepts as literals.

What is removed is `\A...\z`, `\G`, `\Q...\E`, Perl's `\x{...}` and Unicode properties, which is Perl and Ruby syntax, and it is enriched for the anchored validator shapes the matched comparison rests on: 23.7% of what is dropped is anchored against 17.2% of what is kept. The direction of that bias is not obvious: the read-to-be-read populations lose the larger share, and losing vulnerable patterns from those would understate the very gap we report. Assuming either direction is guessing, so we measure it. We translate what can be translated without guessing (`runner/dialect.py`: `\A`→`^`, `\z`→`\Z`, `\Q...\E` to an escaped literal, named groups to Python's spelling), decline what cannot (`\G` anchors to the previous match and has no Python equivalent; `\p{...}` would have to be approximated), and report the recovered population as a robustness row alongside the original.

Two registries cannot backtrack at all. `crates.io` and `godoc` target RE2 and the Rust `regex` crate, which are linear-time by construction. ReDoS is not a property their patterns can have, whatever their shape, so screening them as though they ran under a backtracking engine is a category error. 23,941 patterns appear in one of the two; of those, 20,658 appear in no backtracking registry at all, and those are the ones the robustness run excludes. (A pattern can be published to several registries, so this is a count of distinct patterns rather than the sum of the two registry columns.)

Neither correction moves the number. Normalisation recovers 17,151 of the 26,900 dropped production patterns (64%). What stays out is mostly not exotic syntax: the largest group, 4,588 patterns, is malformed or truncated by the static extraction that produced the corpus (template fragments rather than regular expressions) followed by `\G` and Unicode properties, which we decline rather than approximate. Screening the recovered population takes anchored production from 8.9% to 9.1%; dropping the linear-engine registries takes it to 8.3%; doing both gives 8.2%. The whole

range is 8.2–9.1%, well inside the interval on any one of them, and the models at 9.8% remain indistinguishable from every variant. The dialect filter and the engine mismatch are real defects in the instrument, and they are not what produces the result.

Recall, measured per population. Agreement between two static approximations is not calibration, so we pair an independent detector with a dynamic oracle. The detector is `weideman-RegexStaticAnalysis`, from the artifact of Davis et al. [Davis et al., 2019] and the same one their ecosystem study used: a different analysis, a different language, a different author. When it calls a pattern vulnerable it emits an exploit string, and we then build that input at growing pump counts and time *CPython's own matcher* on it under a hard timeout. A pattern is ground-truth vulnerable when matching either fails to finish or grows measurably super-linearly in the input length. That is a fact about the engine this paper screens with, not a second opinion.

Population	sampled	unanalysable	caught/confirmed	recall (%)	95% CI	agreement (%)	med. len
Stack Overflow	240	80	25/27	92.6	[77, 98]	67.9	27
RegexLib	240	89	33/36	91.7	[78, 97]	65.0	67
Re(gEx|DoS)Eval gold	219	60	19/22	86.4	[67, 95]	65.8	46
Production code	240	46	16/20	80.0	[58, 92]	74.2	23
NL-RX-Synth	240	56	16/21	76.2	[55, 89]	62.1	24
Model outputs	240	64	11/15	73.3	[48, 89]	55.4	55
KB13	209	110	2/3	66.7	[21, 94]	52.6	15

Table 6: Screen sensitivity per population. *Unanalysable* counts patterns the independent detector skipped or timed out on, which are neither evidence for nor against. *Caught/confirmed* is our screen's hits over the patterns the detector flagged *and* a timing measurement then confirmed blow up under CPython. Recall is measured against what the detector finds, so it is a relative sensitivity rather than an absolute one, which is what the cross-population comparison needs, since the question is whether the instrument is equally blind everywhere and not how blind it is. The intervals are wide because dynamically confirmed vulnerabilities are scarce in a sample of a few hundred; they are given so that the comparison is read at the precision the counts support.

Recall runs 67–93% across the seven populations, and the intervals overlap almost everywhere: dynamically confirmed vulnerabilities are scarce in a sample of a few hundred, and KB13 yields three. So this does not resolve a fine ordering of sensitivities, and we do not claim one.

What it does resolve is magnitude. The direction comes first, because the differential runs *towards* the explanation that would deflate this section, not away from it. If the ordering were manufactured by the screen being blinder in production code than in showcase validators, recall would have to be lower in production. It is: production code sits at 80.0% against `regexlib.com`'s 91.7% and Stack Overflow's 92.6%. The hypothesis makes a correct prediction about the sign.

It is wrong about the size, which is what settles it. Dividing each measured rate by the recall behind it gives a first-order correction. On the anchored populations that turns 20.1%, 17.3%, 13.4%, 9.8% and 8.9% into 21.9%, 18.7%, 15.5%, 13.4% and 11.1%. Every rate rises, because the screen is a lower bound everywhere. The gap between the most vulnerable read population and production code narrows from 11.2 points to 10.8 (the differential eats 0.4 of a 11.2-point gap) and the ordering does not change. A miss rate that would have to close eleven points closes four tenths of one.

The same correction moves the models off production code. It also costs us the other half of the pairing, and by more than the anchoring correction did. Model outputs have the second-lowest

recall in Table 6 at 73.3%, so correcting them moves them further than it moves production: 9.8% to 13.4% against 8.9% to 11.1%. The distance between the two goes from 0.9 points to 2.2, and the models land nearer the corrected gold set at 15.5% than the corrected production code they are paired with in Table 5.

We do not think that breaks the pairing, because the interval on the models' 73.3% recall is [48, 89] and a correction that noisy cannot carry a 2-point conclusion. But it is the same kind of disclosure as the eight tenths of a point the anchoring correction cost, it is larger, and a reader should not have to compute it from two adjacent paragraphs. We would not put the corrected figures in a table (recall here is measured against what one detector finds, so it is relative rather than absolute, and the counts behind it are small), but they belong in the text.

Three limitations belong with it. Recall is conditioned on what the independent detector flags, so it estimates the screen's sensitivity without bias only if the *detector's* sensitivity is itself population-independent, which is a version of the assumption this section exists to test, moved up one level. The timing oracle keeps this from being a two-detector agreement study, since a confirmation is a measured fact about CPython rather than a second opinion, but it does not remove the conditioning. Relatedly, the detector could not analyse between 46 and 110 of each 240-pattern sample, and those are neither evidence for nor against; ground truth therefore covers the patterns two tools can both reason about, and two shape-based analyses share blind spots, so what both miss is structurally invisible here. Agreement between our screen and the detector runs 53–74%, far below either one's recall, because the detector flags a good deal that does not blow up in CPython: `^[a-z]+[a-z]*$` is quadratic in principle and instant in practice. That gap is the reason this section uses a timing measurement as the arbiter rather than a second opinion.

Relation to prior work, and a caveat on the split. The corpus's companion ReDoS study reports that LLM-generated regexes skew toward polynomial ReDoS [Siddiq et al., 2024b]. We do not reproduce this in our models: pooled, we observe 5.3% exponential against 3.8% polynomial. The cross-corpus run locates the skew elsewhere. In the anchored production sample, polynomial exceeds exponential 253 to 105, while this corpus's gold answers are near even at 34 to 38.

That comparison is weaker than it looks. Our exponential-versus-polynomial split is only partly a claim about growth order. The structural pass assigns it from the shape it matched, which is principled; the empirical pass assigns it from *which probe length first timed out*, which is a threshold, not a measurement. Patterns flagged only empirically therefore carry a degree label our instrument did not really establish.

How much of the split that affects, we can say for one population. Among the generations that satisfied every test, on both corpora, *every* vulnerable verdict came from the structural pass and none from the empirical one, so for those the degree is shape-derived rather than threshold-derived. We have not established the same for the reference sets or for the wild populations, where patterns are longer and more of them fall outside the structural pass's grammar. The subcategory counts are also small. We report the discrepancy with prior work as an observation that wants a replication under a detector that measures growth order directly, and draw nothing from it.

4.6 Comparison against joint benchmarks in other domains

Because USABLE carries a semantic-equivalence conjunct that the joint benchmarks of Section 2.1 do not, comparing it directly to their figures would understate us. We therefore also report *correct-and-secure*: correctness conjoined with safety, omitting the equivalence term, which is the construction those benchmarks use.

Benchmark	tasks	functional (%)	joint (%)	vuln.|correct (%)
This work (regex, ReDoS)	450	42	39	7
BaxBench [Vero et al., 2025]	392	62 (best)	—	≈50
SecureAgentBench [Chen et al., 2025]	105	—	15.2 (best), 9.2 (mean)	—
DualGauge [Patir et al., 2025]	154	>50	<12	—

Table 7: Our correct-and-secure rate against joint correctness-security results from other domains. Task types, vulnerability classes and evaluation methods all differ substantially. The comparison is one of magnitude and of the ratio between functional and joint success, not of models or difficulty.

We do not reproduce the pattern. BaxBench reports that roughly half of functionally correct backends are exploitable [Vero et al., 2025]. SecureAgentBench reports 15.2% correct-and-secure for their strongest agent, against substantially higher functional correctness. DualGauge reports secure-pass@1 below 12% while functional pass@1 exceeds 50%. In each case the security criterion removes a large majority of functionally correct output.

In our setting it removes 7.4%.

The one prior correct-and-secure figure on *this* corpus is the original study's, over T5, Phi-1.5 and GPT-3.5-Turbo [Siddiq et al., 2024a]. We do not put it in Table 7. Their generation, extraction and dialect handling differ from ours in ways we did not control for, so a difference between their number and ours would not be attributable to the two model populations, which is the only comparison that would be interesting. Anyone wanting that comparison should re-run their harness on the current models rather than read it off the two papers.

We have three explanations for the divergence. One is testable on this data, and the test is in Section 4.7; the other two are not:

1. **Attack surface.** A regular expression is a single expression with one failure mode of interest. A backend application has many independently exploitable components, so the probability that at least one is vulnerable compounds with program size.

2. **Vulnerability class.** ReDoS is a structural property of the pattern, plausibly well represented in training data as a recognised anti-pattern. CWE-classified defects in application code are more diverse and more context-dependent.

3. **Task difficulty ceiling.** Our functional pass rate is itself low (38.0–46.5%), so the correct subset may be biased toward simpler tasks whose solutions have less structural room for catastrophic backtracking.

The third is the one that would deflate the finding, and it is the one we can test. The cross-corpus form of the test is available first: if the correct subset is safe because it is simple, StructuredRegex's correct patterns should be simpler still, being twenty points easier, and therefore safer. They are neither: they are *longer* (median 33 characters against 28) and carry *more* quantifiers (median 4 against 3) while being twice as ReDoS-prone, and Section 4.7 takes that further.

But that test compares two corpora whose targets were authored by completely different processes, and Section 4.7 then argues at length that the difference in authorship is what drives the vulnerability gap. It therefore cannot separate "the correct subset is biased toward simple tasks" from "the two corpora differ". The within-corpus form can, because authorship is held fixed. We stratify this corpus's tasks by how many of the eleven models solved them (the corpus's own measure of how easy a task turned out to be) and measure vulnerability among correct patterns inside each stratum.

The selection story survives this, weakly, and against us. On tasks that six or more models solved, 7.1% of correct patterns are vulnerable; on tasks only one to five solved, 10.7%. The point estimate runs the direction the hypothesis predicts, so part of our headline rate does look like a selection effect. Resampled over tasks the difference is -3.6 points with a 95% interval of $[-12.2, +3.8]$, which crosses zero, so this corpus cannot establish it.

What the stratification does establish is a bound, and that is the useful part. Even on the tasks only one to five models could solve, vulnerability among the patterns they did get right is 10.7%, and not the roughly 50% that BaxBench reports for backend code. The difficulty ceiling is real enough to be worth stating and far too small to be the explanation.

The first two we cannot separate here. Both predict that the regex domain looks safer than backend code, and neither predicts anything about the gap between our two corpora, so nothing in this study distinguishes them.

So the claim that generalises is weaker than "correct code is often insecure," and more useful. *The size of the correctness-to-security penalty depends on the domain and has to be measured there.* A composite that is not decomposed will not tell you this, because it can report a gap of about the expected size for an entirely unrelated reason (Section 4.3). An undecomposed joint metric will mislead you exactly there, where its aggregate agrees with the literature and its components do not.

4.7 Replication on a second corpus

Every figure above comes from one corpus, so the 7.4% is a property of Re(gEx|DoS)Eval until shown otherwise. The two criteria that never consult a reference can be computed anywhere that supplies test strings, and StructuredRegex [Ye et al., 2020] supplies them: each instance carries positive and negative example strings alongside its description. Its own targets are in a prefix DSL and we never read them, so this run has no DFA-EQ and no EXACT term, which suits the argument of this paper.

We collected one sample per task from the same eleven pinned models under the same configuration, 622 of the 629 test instances (seven ship no positive or negative strings, leaving PASS@k undefined). Table 8 reports the 513 tasks every model answered. Section B records why that number is not 622.

The qualitative result replicates and the quantitative one does not. Working patterns are exploitable at a material rate on both corpora, which is the finding. But the rate is **16.5%** here against 7.4% on Re(gEx|DoS)Eval, and the obvious deflationary reading, that harder problems produce worse patterns, is ruled out by the direction of the difficulty gap: StructuredRegex is *easier*, by roughly twenty points of PASS@1.

Which shape, and why a grammar produces it. The structure of the correct patterns says where the extra vulnerability comes from, and it is not where one would guess. StructuredRegex's correct patterns are longer (median 33 characters against 28) and carry more quantifiers (median 4 against 3), but they are *less* nested: a quantifier applied to a group (the `(a+)+` shape everyone pictures when they hear ReDoS) appears in 18.6% of them against 34.7% on Re(gEx|DoS)Eval, and their median group nesting depth is zero. The corpus with fewer of the canonical exponential shapes is the corpus with twice the vulnerability rate.

What it has instead is repetition in *sequence*, and the screen's own reasons say so. Of the vulnerable correct patterns on Re(gEx|DoS)Eval, 236 are flagged for a quantifier wrapping a quantified group and 154 for two quantifiers in sequence over overlapping character sets, 40% adjacent. On StructuredRegex the shares invert: 202 nested against 355 adjacent, 64% adjacent. In

Model	PASS@1	95% CI	VULNERABLE@1	correct-and-secure	VULNERABLE |correct
`claude-sonnet-5`	66.1	[61.9, 70.0]	14.2	56.3	14.7
`gpt-5.6-sol`	65.5	[61.3, 69.5]	14.0	55.8	14.9
`claude-opus-5`	61.6	[57.3, 65.7]	14.4	51.9	15.8
`gpt-5.6-terra`	62.2	[57.9, 66.3]	16.0	51.9	16.6
`qwen3.6-max-preview`	60.6	[56.3, 64.8]	13.6	51.7	14.8
`gpt-5.6-luna`	59.8	[55.5, 64.0]	15.0	50.1	16.3
`gemini-3.1-flash-lite`	60.0	[55.7, 64.2]	16.6	48.7	18.8
`kimi-k3`	58.9	[54.6, 63.0]	17.2	48.7	17.2
`qwen3.6-plus`	55.9	[51.6, 60.2]	15.6	46.4	17.1
`glm-5.2`	55.9	[51.6, 60.2]	17.9	45.0	19.5
`deepseek-v4-flash-0731`	51.7	[47.3, 56.0]	15.8	43.5	15.8
Pooled	59.8	—	15.5	50.0	**16.5**

Table 8: StructuredRegex, one sample per task, on the 513 instances answered by all eleven models. Percentages. Intervals are Wilson binomial, which is appropriate here in a way it is not in Table 1: this is one sample per task, so there is no within-task resampling for a bootstrap to exploit. The last column is the share of functionally correct patterns that are ReDoS-vulnerable, the same quantity this paper reports as 7.4% on Re(gEx|DoS)Eval.

	Re(gEx|DoS)Eval	StructuredRegex
Tasks	450	513
Reference used in scoring	yes (DFA-EQ)	no
PASS@1, range	30.0–41.4%	51.7–66.1%
VULNERABLE@1, range	7.4–10.7%	13.6–17.9%
VULNERABLE |correct, pooled	**7.4%**	**16.5%**

Table 9: The two reference-independent criteria across both corpora, at @1 in each so the comparison is like for like.

absolute terms the nested count barely moves between corpora while the adjacent count more than doubles. The extra vulnerability is entirely the polynomial family: `\s*\s*`, `[a-z]+[a-z0-9]*`.

Which is what its targets are built out of. A grammar over repetition, optionality and concatenation produces adjacency by construction, and Section 4.5.1 found the identical signature in NL-RX-Synth, the other grammar-built corpus here: 891 of its 2,057 vulnerable targets are flagged for that same shape, and the population screens at 35.2%. Two grammar-generated corpora, built by different authors for different purposes, over-represent the same family.

We think the reading is this. A grammar samples compositions uniformly and has no reason to avoid adjacency, because nothing downstream of it runs the output; a human writing a validator writes one nested quantifier and stops. So the correctness-to-security penalty depends not only on the domain but on how the corpus's targets were *authored*, which is a sharper and more uncomfortable claim than the one we set out with.

Section 4.6 argues from BaxBench and SecureAgentBench that the correctness-to-security penalty is domain-dependent. That argument compares across languages and vulnerability classes, and a reader is entitled to discount it. The same conclusion now holds between two corpora of the same task, in the same language, under the same screen and the same eleven models, with the effect size differing by a factor of 2.2. We regard this as the stronger form of the claim, and it cuts against our own headline number.

The ordering of models is also not preserved. `claude-sonnet-5` leads here and sits mid-table on Re(gEx|DoS)Eval; `kimi-k3` leads there and sits eighth here. Section 5.4 supports bands, not a ranking, on single-corpus evidence, and this is the direct test of that caution.

4.8 Cost

Model	USABLE@3 (%)	cost/request ($\$\times 10^{-6}$)	total ($)
`deepseek-v4-flash-0731`	19.8	25.6	0.03
`gpt-5.6-luna`	18.5	42.9	0.06
`gemini-3.1-flash-lite`	17.1	90.2	0.12
`qwen3.6-plus`	19.8	120.6	0.16
`glm-5.2`	18.7	158.2	0.21
`qwen3.6-max-preview`	21.6	387.9	0.52
`gpt-5.6-terra`	18.7	406.2	0.55
`claude-sonnet-5`	18.0	932.4	1.26
`kimi-k3`	23.8	1328.1	1.79
`gpt-5.6-sol`	20.9	2107.6	2.85
`claude-opus-5`	20.8	2514.2	3.39

Table 10: Cost per *request*, measured from per-request billing returned by the serving layer rather than estimated from list prices. A task costs three of these at $n = 3$: `claude-opus-5` at 2514.2×10^{-6} over 1,350 requests gives the $3.39 total.

The extremes differ by a factor of 98 in price and 1.1 percentage points in USABLE@3, in favour of the more expensive model: `deepseek-v4-flash-0731` scores 19.8% and `claude-opus-5` 20.8%. Before the estimator fix of Section B the gap was 3.2 points in the same direction, so the correction shrinks it rather than inverting it, and 1.1 points is well inside what we can resolve (Section 5.3), so on this task the two are indistinguishable and the cheaper one does not win.

4.9 Refusals

A content filter blocked `claude-opus-5` on 29 requests, 2.1% of its calls, spread across five tasks. Four have some security flavour: strings that exclude a single quotation mark, a six-character alphanumeric password, hex byte sequences, and SQL update and insert statements. The fifth is *"Matches a file extention."* No other model refused anything.

The refusals were not deterministic. Several tasks came back refused on one sample and answered on the other two, under identical settings. You can only see that at $k > 1$. A single-sample protocol would have logged a flat failure and moved on. We count refusals as their own failure category, not as wrong answers (Section A).

4.10 Inference-time compute

On the corpus's simplest task (*"Matches exactly 1 numeric digit (0-9)."*), `qwen3.6-max-preview` emitted 1,571 hidden reasoning tokens before answering `^[0-9]$`; with reasoning disabled it produced the identical answer in 10 tokens—100 times cheaper. Asking that model for *low* reasoning effort produced *2,375* reasoning tokens, more than the default. Effort controls are not reliably monotone.

A reasoning-enabled comparison on 12 tasks across five models yielded per-request costs 15.7× higher; the accuracy signal is uninterpretable at that sample size (±14 percentage points per cell).

5 Measurement Validity

5.1 The reference standard contains errors

A benchmark's answer key is a set of labels, and labels have an error rate. An audit of ten widely used test sets found a mean 3.3% label-error rate [Northcutt et al., 2021], enough in several cases to reverse the ordering of models the benchmark was being used to compare. The corpus here differs in a way that helps: a label is a regular expression, so a disputed label can be settled by exhibiting a string on which the reference and the description disagree, rather than by a second opinion. Every adjudication below rests on such a string.

First we ruled out a loading fault. 449 of 450 reference patterns satisfy their own tests, so the corpus semantics and dialect are configured correctly. The real defect is quieter. A reference can pass its own tests and still fail to denote what its description says, whenever the tests never exercise the difference.

We drew a seeded random sample of 60 cases in which a model *passed every test* yet received a DIFF verdict, and adjudicated 14 in detail. The 14 are *the first 14 in seed order whose witness string is ASCII*: the non-ASCII cases are a known and separate artifact, and adjudicating them would have been adjudicating Python's Unicode defaults rather than either party's regular expression.

Adjudicated cause	count	share
Reference pattern is incorrect	5	36%
Description underspecifies the disputed property	6	43%
Model output is incorrect	3	21%

Table 11: Adjudication of 14 sampled DIFF verdicts on test-passing outputs. Per-case reasoning is released.

Separately, 19 of the 60 sampled disagreements (32%) differ only on non-ASCII input. Python's `\d` is Unicode-aware and `[0-9]` is not, so the two are formally distinct languages and practically

the same thing.

Because the adjudicated 14 were drawn from the ASCII-witness cases only, none of them is among those 19, and the two effects compose without double-counting. The model is at fault in 3 of 14 ASCII-witness cases, and 41 of the 60 sampled cases have an ASCII witness. What follows depends on how the non-ASCII cases are treated, and the two defensible readings differ by six points, so we give both rather than pick one silently:

- Counting them as DIFF verdicts on which the model is *not* at fault (the `\d`-versus-`[0-9]` distinction is Python's, not the model's) gives $(41/60) \times (3/14) = 14.6\%$.
- Treating them as not real disagreements at all, so that they leave the denominator, gives $3/14 = 21\%$.

We quote the first, roughly **15%**, because a formally distinct language is a real DIFF verdict and dropping it would flatter us. A reader who disagrees should read 21%. Neither number changes the conclusion, which is that the majority of DIFF verdicts are not model errors. The verdict labels in Table 11 map to the released adjudications as `model_right` → reference incorrect, `ambiguous` → description underspecifies, `gold_right` → model incorrect.

Representative cases:

- **Description:** *"It just accepts only positive numbers."* The reference `^\d+([.,]?\d+)?$` admits `0`. The model excluded zero and was scored incorrect.
- **Description:** *"A very simple ISBN validation expression—it just checks for a 10 digit number."* The reference is `^\d{9}[\d|X]$`. The character class contains digit, *pipe* and `X`: an alternation operator written inside a class, where it denotes a literal. The reference therefore accepts `000000000|`, which is the witness separating it from the model's `^\d{9}[\dX]$`.
- **Description:** *"...make sure commas are in the rite place (if present)."* The reference

  ```
  ^\$?\d{1,3}(,?\d{3})*(\.\d{1,2})?$
  ```

  makes the comma optional, admitting `0,000000` and negating the stated purpose. The model enforced comma placement and was scored incorrect.
- **Underspecification:** *"Matches any single upper- or lower-case letter."* The reference `^[a-zA-Z]$` requires a whole-string match. A candidate `[A-Za-z]` requires only occurrence. The description does not distinguish the two. We tested whether anchoring alone explains failures generally: it accounts for approximately 6%, so the effect is diffuse rather than a single correctable convention.

Consequence. DFA-EQ and USABLE are lower bounds on model correctness by an unmodelled margin. PASS and VULNERABLE are unaffected, since neither consults the reference.

Limitation of this analysis. Fourteen cases, judged by us, on our own benchmark. That is a weak basis for a precise number and we do not want it cited as one. We release the reasoning for each case, and the unadjudicated cases keep an empty verdict field so someone else can fill them in. Anyone quoting 15% should re-derive it from a larger sample with annotators who have no stake in the answer. We report it because the direction is not in doubt, and because the direction is what changes how the composite should be read.

5.2 The composite rewards putting an answer beyond checking

DFA-EQ counts a verdict the engine could not settle as a failure. USABLE does the opposite: its third conjunct excludes only a *proven* DIFF, so any comparison the engine cannot answer is scored as a pass. Each convention is defensible alone (one is a lower bound that cannot flatter, the other isolates the model from the engine's reach) but together they hand out credit for being unanswerable, and between 78 and 133 of each model's scored tasks are unanswerable.

We expected the mechanism to be backreferences. Equivalence really is undecidable on them, so a model reaching for one would be scored *more* usable for having put its answer beyond checking, and that is a tidy story about model behaviour. It is also almost entirely wrong. Of the 471 task-credits that rest on an unsettled verdict, the split is 451 UNSUP, 20 UNDEC: the undecidability theorem accounts for twenty of them, and the other 451 are UNSUP, our engine declining a construct.

That is worse, not better. The constructs are ordinary: a lookahead containing an anchor (`^(?=.{1,253}$)...`), `\A` where the corpus expects `^`, an inline flag group, a pattern whose automaton exceeds a state budget. None is exotic and none says anything about the model. Worse still, the declined side is sometimes the *reference*: when the answer key uses a construct the engine will not take, every model is credited on that task no matter what it wrote. The composite's leniency tracks the coverage of our equivalence engine, and a reader would reasonably have assumed it tracked the models.

The obvious test misses this entirely. Correlating a model's unsettled count against its USABLE@3 returns nothing (Pearson -0.15), because unsettled credit and genuine equivalence skill push the composite in opposite directions and a correlation on the sum nets them out. So we decompose. Table 12 recomputes USABLE@3 under the stricter rule that the verdict must be *proven* EQ.

Model	undec.	USABLE@3	proven-EQ only	tasks	share of USABLE (%)
`kimi-k3`	78	23.8	13.2	47	44.8
`qwen3.6-max-preview`	94	21.6	12.7	40	41.2
`gpt-5.6-sol`	133	20.9	8.7	55	58.5
`claude-opus-5`	104	20.8	10.6	44	48.9
`deepseek-v4-flash-0731`	93	20.0	10.9	41	45.6
`qwen3.6-plus`	99	19.8	10.2	43	48.3
`glm-5.2`	86	18.7	9.6	41	48.8
`gpt-5.6-terra`	100	18.7	9.8	40	47.6
`gpt-5.6-luna`	112	18.5	8.5	45	54.2
`claude-sonnet-5`	97	18.0	9.8	37	45.7
`gemini-3.1-flash-lite`	95	17.1	8.7	38	49.4
pooled	—	—	—	471	**48.4**

Table 12: What the unsettled-verdict convention is worth. "Proven-EQ only" requires a demonstrated equivalence rather than the absence of a demonstrated difference; the final columns are the tasks that separate the two. Denominators are Table 1's, so the two tables agree.

48% of all USABLE@3 credit (471 of 974 task-credits) rests on a verdict the engine never reached, and the share tracks the unsettled count closely across models ($r = +0.82$) even though the composite itself does not. Removing the credit reorders the field. `gpt-5.6-sol` has the most unsettled comparisons of any model at 133; it has the third-highest USABLE@3 and, on proven equivalence, ties for third-lowest, a drop of six places, or seven depending on how the tie with `gemini-3.1-flash-lite` at 8.7% is broken.

We are not proposing the stricter rule: it would charge a model for a theorem in twenty cases and

for our parser's limits in 451, which is worse than what we have. But nearly half of what USABLE certifies, it certifies by default, and the default fires on a property of the measuring instrument rather than of the thing measured: a third independent reason to distrust the composite.

5.3 Paired inference

Independent binomial confidence intervals are the wrong estimator for this design, and they are the default in adjacent evaluation work. All eleven models answer the same items, so task difficulty is a shared random effect. Treating the systems as independent samples charges every one of them separately for the same variation.

A paired bootstrap fixes this, and the fix is not ours: the procedure comes from the significance-testing literature [Berg-Kirkpatrick et al., 2012], following the machine-translation protocol [Koehn, 2004], and it is the recommended practice for exactly this design [Dror et al., 2018]. Resample tasks with replacement [Efron and Tibshirani, 1994], score every model on each resample, and form the per-pair differences. Task difficulty cancels, and only disagreement between the two systems contributes. With $B = 10{,}000$ resamples and a fixed seed:

Procedure	pairs resolved at 95% (of 55)
Independent binomial intervals	1
Paired bootstrap, USABLE@3	9
Paired bootstrap, PASS@3	16
Paired bootstrap, VULNERABLE@3	9

Table 13: Resolvable pairwise comparisons under independent versus paired inference on identical data.

The correction is not marginal, it changes the number of defensible comparisons on USABLE@3 9-fold. We flag it because independent intervals look like the default in adjacent evaluation work, and the design they are applied to, all systems answering all items, is close to universal.

Even corrected, the middle of the table does not separate. For example `qwen3.6-max-preview` versus `gpt-5.6-sol` yields $+1.2\%$ with a 95% interval of $[-2.6\%, +4.8\%]$.

5.4 Discriminative power of the corpus

There is a structural reason behind this. Over the 421 tasks every model answered, **62% produce the same usable@3 outcome for all eleven**. On PASS@3 it is 56%, on VULNERABLE@3, 78%. Only 162 of 421 tasks carry any USABLE@3 signal at all.

For calibration: resolving a 3-percentage-point difference at $p \approx 0.2$ with independent intervals requires roughly 2,800 items, more than the full 762-task corpus contains. Paired inference reduces this requirement substantially but cannot manufacture signal from items on which every system agrees. Benchmark construction for system comparison should therefore optimise for discriminative yield, not corpus size.

6 Threats to Validity

Contamination. The corpus predates every model we evaluated and is public, as are the forum posts it was built from. Some of what we measured may be memorisation. This has to be measured per benchmark rather than assumed negligible [Sainz et al., 2023], and we cannot measure it: we

have no held-out private task set, so we cannot put a bound on it. This is the largest unaddressed threat in the paper.

Corpus-local effect sizes. Section 4.7 replicates PASS and VULNERABLE on a second corpus and finds the vulnerability rate among correct patterns differs by a factor of 2.2. Every effect size in this paper should therefore be read as a property of Re(gEx|DoS)Eval rather than of natural-language-to-regex generation. DFA-EQ and EXACT remain single-corpus: replicating those needs a second corpus with a reference in standard syntax, and neither KB13, NL-RX nor StructuredRegex supplies one. A further hazard we avoided rather than solved: KB13 and NL-RX are written in a DSL where `&` is intersection and `~` is negation, both of which Python's `re` accepts silently as literals. We excluded the affected patterns instead of mistranslating them: NL-RX-Synth goes from 10,000 rows to 9,648 distinct targets to the 5,840 screened, and KB13 from 824 rows to 732 distinct to 532, with both counts stated over distinct targets so they reconcile.

Single prompt. Every result here is conditional on one unoptimised prompt. Section 5.1 shows how often the task descriptions leave the anchoring convention unstated, so sensitivity to phrasing is plausibly large. We did not measure it.

Sampling control. We do not set temperature (Section 3.5), so it varies by provider default. We checked that outputs still vary across samples. We did not equalise the sampling.

Estimator degeneracy. At $n = k = 3$ the PASS@k estimator reduces to any-of-3.

Coverage asymmetry. Nine models cover all 450 tasks. `kimi-k3` covers 447, having exhausted our budget mid-run, and `claude-opus-5` covers 444, losing five tasks to content-filter refusals and one to replies that were nothing but a code fence (Section A). Tasks that came back with fewer than k samples are excluded from that model's @k estimate rather than scored on what arrived (Section 3.4), so the denominators in Table 1 are 432–450.

Screening, not proof. VULNERABLE is a lower bound (Section 3.4).

Single corpus. We do not test whether the observed banding survives corpus change.

Self-adjudication. See Section 5.1.

7 Recommendations for Evaluation Design

1. **Partition metrics by reference dependence, and decompose any composite that spans the partition.** Metrics evaluated by execution against inputs (PASS, VULNERABLE) are unaffected by reference-set defects. Metrics evaluated against a human answer key inherit its error rate. A composite conjoining both can report a large gap that is almost entirely attributable to the unreliable term, in our case 87% of it (Section 4.3), while appearing to be a statement about the reliable ones.

2. **Sample and adjudicate disagreements.** It costs hours. In our case it changed the conclusion.

3. **Use paired inference when all systems answer all items.** Table 13.
4. **Report discriminative yield**, not only corpus size (Section 5.4).
5. **Pin the serving provider and verify post hoc.** Record what served each request, not only what was requested (Section 3.3).
6. **Instrument the scorer with controls** that must pass and must fail (Section 3.4).
7. **Release raw generations.** Anyone can then overturn a scoring decision without buying the inference again.
8. **Treat an effect size as corpus-local until replicated.** Our reference-independent gap is 7.4% on one corpus and 16.5% on another under an identical screen (Section 4.7). Reference-independence buys freedom from answer-key error. It does not buy generality.
9. **Check that a partial run is not a biased run.** When a model loses tasks to infrastructure, score the remaining models on the tasks it lost. Ours were 8.1 points harder (Section B), which would have flattered the affected model rather than penalised it. Check separately that the *estimator* is defined on the partial data: ours was not, and credited every short task to whichever model lost it (Section B).
10. **Generate every table from committed data, and let the build fail.** The reviewer of an earlier draft of this paper found nine arithmetic inconsistencies. Every one of them was in a table we maintained by hand or in a sentence quoting one, and none was in a table generated from the scores. The fix is mechanical rather than editorial: emit all of them, keep the numbers the prose states in macros generated alongside, and have continuous integration rebuild and diff.
11. **Calibrate a screen per population before comparing populations.** A detector's false-negative rate is a property of the patterns it is pointed at. Reporting it as a single caveat is adequate for one population and lets a differential miss rate masquerade as a finding across several (Section 4.5.2).

8 Conclusion

Regular expressions are a domain where correctness is partly decidable, where the output runs, and where a test-passing answer can still be a denial of service. We measured eleven current models on all three axes over 450 tasks, and then took the one axis that needs no model at all and applied it to five further populations of regular expressions.

That extension is where the strongest result is. Restricted to a common pattern shape, ReDoS prevalence across six populations is ordered by whether a pattern was ever executed. Reusable library entries screen at 20.1%, forum answers at 17.3%, and this corpus's own human reference answers at 13.4%. Regular expressions extracted from shipped packages sit at 8.9%, and the models sit with them at 9.8%, a difference of 0.9 points that does not resolve (two-proportion z, $p = 0.20$). Calibrating the screen against an independent detector and a dynamic timing oracle moves every population in the same direction and leaves the ordering intact. The variable that predicts catastrophic backtracking is exposure to execution, and authorship by a human or a model predicts very little.

The correctness-to-security penalty in this domain is small. 7.4% of functionally correct patterns carry a ReDoS vulnerability, against roughly 50% of functionally correct backends in comparable work on backend code, so requiring both costs 2.9 percentage points here and around half the field there. The same eleven models under the same screen on StructuredRegex give 16.5% on tasks twenty points easier (Section 4.7), which fixes the size of the penalty as a property of the corpus. The general claim that survives is that the penalty is domain-dependent and has to be measured in the domain it is claimed for. Over those same eleven models the field spans 6.7 points across a 98× range in inference price, and price does not order it.

The methodological result is that a metric which compares against a human answer key inherits that key's error rate. We adjudicate a random sample of cases where a model passed every test and was scored semantically different, and find the model clearly at fault in roughly 15% of them; the rest are incorrect reference patterns and prompts that never specified the property in dispute. A composite conjoining correctness, safety and equivalence draws 87% of its gap from that term, and awards free credit wherever equivalence is undecidable, which is a property of the parser rather than of the answer. We recommend that evaluations separate reference-dependent from reference-independent criteria, report them separately, and give them different evidentiary weight. Section C records the order in which we established the results above, for readers who want the audit trail rather than the findings.

Acknowledgements

Scoring is performed by `regexbench` [Plicara Labs, 2026a], prior work by the present authors, used here as a pinned dependency and not contributed by this paper. The corpus is Re(gEx|DoS)Eval [Siddiq et al., 2024a], used under its original terms and not redistributed.

References

Taylor Berg-Kirkpatrick, David Burkett, and Dan Klein. An empirical investigation of statistical significance in NLP. In *Proceedings of EMNLP-CoNLL*, pages 995–1005, 2012. URL `https://aclanthology.org/D12-1091/`. The paired bootstrap procedure adopted in this paper.

Masudul Hasan Masud Bhuiyan, Berk Çakar, Ethan H. Burmane, James C. Davis, and Cristian-Alexandru Staicu. SoK: A literature and engineering review of regular expression denial of service (ReDoS). In *Proceedings of the 20th ACM Asia Conference on Computer and Communications Security (AsiaCCS)*, 2025. doi: 10.1145/3708821.3733912.

Junkai Chen, Huihui Huang, Yunbo Lyu, Junwen An, Jieke Shi, Chengran Yang, Ting Zhang, Haoye Tian, Yikun Li, Zhenhao Li, Xin Zhou, Xing Hu, and David Lo. SecureAgentBench: Benchmarking secure code generation under realistic vulnerability scenarios, 2025. 105 repository-level tasks. Best agent (SWE-agent with DeepSeek-V3.1) attains 15.2% correct-and-secure; mean across agents 9.2%.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. Source of the unbiased pass@k estimator used in Section 3.4.

James C. Davis, Christy A. Coghlan, Francisco Servant, and Dongyoon Lee. The impact of regular expression denial of service (redos) in practice: An empirical study at the ecosystem scale. In *Proceedings of ESEC/FSE*, pages 246–256, 2018. doi: 10.1145/3236024.3236027.

James C. Davis, Louis G. Michael IV, Christy A. Coghlan, Francisco Servant, and Dongyoon Lee. Why aren't regular expressions a lingua franca? an empirical study on the re-use and portability of regular expressions. In *Proceedings of ESEC/FSE*, 2019. Artifact: `https://github.com/VTLeeLab/LinguaFranca-FSE19`.

Rotem Dror, Gili Baumer, Segev Shlomov, and Roi Reichart. The hitchhiker's guide to testing statistical significance in natural language processing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1383–1392, 2018. URL `https://aclanthology.org/P18-1128/`.

Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall/CRC, 1994.

James Kirrage, Asiri Rathnayake, and Hayo Thielecke. Static analysis for regular expression denial-of-service attacks. In *Network and System Security (NSS)*, 2013.

Philipp Koehn. Statistical significance tests for machine translation evaluation. In *Proceedings of EMNLP*, pages 388–395, 2004. URL `https://aclanthology.org/W04-3250/`.

Nate Kushman and Regina Barzilay. Using semantic unification to generate regular expressions from natural language. In *Proceedings of NAACL-HLT*, pages 826–836, 2013. URL `https://aclanthology.org/N13-1103/`. The KB13 corpus.

LessWrong. Not pinning your openrouter provider might invalidate your research. `https://www.lesswrong.com/posts/KsyoSAyBRXtwzSugg/`, 2025. Reports that 31 of 32 surveyed AI-safety codebases using OpenRouter did so without pinning the serving provider.

Nicholas Locascio, Karthik Narasimhan, Eduardo DeLeon, Nate Kushman, and Regina Barzilay. Neural generation of regular expressions from natural language with minimal domain knowledge. In *Proceedings of EMNLP*, pages 1918–1923, 2016. URL `https://aclanthology.org/D16-1197/`. Introduces NL-RX; establishes DFA-equivalence as the standard correctness criterion for this task.

Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947.

Anders Møller. dk.brics.automaton — finite-state automata and regular expressions for java. `https://www.brics.dk/automaton/`, 2021.

Curtis G. Northcutt, Anish Athalye, and Jonas Mueller. Pervasive label errors in test sets destabilize machine learning benchmarks. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2021. Mean 3.3% label-error rate across ten widely used test sets; shows the errors can reverse benchmark rankings.

Rupam Patir, Keyan Guo, Suvadra Barua, Abhijeet Pathak, Dinesh Gudimetla, Jiawei Guo, Hongxin Hu, and Haipeng Cai. DualGauge: Automated joint security-functionality benchmarking of specification-only code generation by LLMs and coding agents, 2025. 154 tasks, 10 models. Reports secure-pass@1 below 12% for all models evaluated while functional pass@1 exceeds 50%.

Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. Asleep at the keyboard? assessing the security of GitHub Copilot's code contributions. In *Proceedings of the 43rd IEEE Symposium on Security and Privacy*, 2022. 1,689 generated programs across 89 scenarios; approximately 40% vulnerable.

Jinjun Peng, Leyi Cui, Kele Huang, Junfeng Yang, and Baishakhi Ray. CWEval: Outcome-driven evaluation on functionality and security of LLM code generation. In *Proceedings of the 2025 IEEE/ACM International Workshop on Large Language Models for Code (LLM4Code)*, 2025. 119 security-critical tasks over 31 CWEs in 5 languages, scored with outcome-driven oracles for functionality and security jointly. Artifact: `https://github.com/Co1lin/CWEval`.

Neil Perry, Megha Srivastava, Deepak Kumar, and Dan Boneh. Do users write more insecure code with AI assistants? In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2023. doi: 10.1145/3576915.3623157.

Plicara Labs. regexbench: Evaluate generated regular expressions — semantic equivalence, correctness, and redos safety. `https://github.com/plicara/regexbench`, 2026a. Version 0.4.1, pinned at commit `ff25e6a5`. Apache-2.0. Prior work by the present authors; used here as a dependency, not contributed by this paper.

Plicara Labs. regexeval-2026: Predictions, scores and analysis. `https://github.com/plicara/regexeval-2026`, 2026b. All raw model responses, scoring code and analysis for this paper.

Oscar Sainz, Jon Ander Campos, Iker García-Ferrero, Julen Etxaniz, Oier Lopez de Lacalle, and Eneko Agirre. NLP evaluation in trouble: On the need to measure LLM data contamination for each benchmark. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023. URL `https://aclanthology.org/2023.findings-emnlp.722/`.

Mohammed Latif Siddiq, Jiahao Zhang, Lindsay Roney, and Joanna C. S. Santos. Re(gEx|DoS)Eval: Evaluating generated regular expressions and their proneness to DoS attacks. In *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, 2024a. doi: 10.1145/3639476.3639757. Introduces the corpus used here (762 tasks) and the pass@k, vulnerable@k, dfa-eq@k and exact@k metrics this paper adopts. Evaluates T5, Phi-1.5 and GPT-3.5-Turbo. Artifact: `https://github.com/s2e-lab/RegexEval`.

Mohammed Latif Siddiq, Jiahao Zhang, and Joanna C. S. Santos. Understanding regular expression denial of service (ReDoS): Insights from LLM-generated regexes and developer forums. In *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension (ICPC)*, 2024b. doi: 10.1145/3643916.3644424. Studies ReDoS in LLM-generated regexes directly; reports a skew toward polynomial vulnerability.

Mark Vero, Niels Mündler, Victor Chibotaru, Veselin Raychev, Maximilian Baader, Nikola Jovanović, Jingxuan He, and Martin Vechev. BaxBench: Can LLMs generate correct and secure backends? In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *PMLR*, 2025. 392 backend tasks (28 scenarios $\times$ 14 frameworks, 6 languages). Reports that roughly half of functionally correct solutions are exploitable.

Valentin Wüstholz, Oswaldo Olivo, Marijn J. H. Heule, and Isil Dillig. Static detection of DoS vulnerabilities in programs that use regular expressions. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2017.

Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. Benchmarking multimodal regex synthesis with complex structures. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020. URL `https://aclanthology.org/2020.acl-main.541/`. 3,520 instances, each with positive and negative example strings. Used here for the second-corpus replication; its targets are in a prefix DSL and are not read.

A Failure taxonomy

Of 14,850 requests, 54 (0.36%) failed and are reported rather than excluded.

Model	Cause	Count
`claude-opus-5`	content-filter refusal	29
`claude-opus-5`	code fence, no pattern inside	7
`kimi-k3`	account spending limit reached mid-run	11
`kimi-k3`	response without resolved provider	3
`kimi-k3`	code fence, no pattern inside	2
`gpt-5.6-sol`	response without resolved provider	1
`gpt-5.6-luna`	code fence, no pattern inside	1
	total	*54*

Table 14: Request failures by cause. Responses lacking a resolved provider are discarded rather than scored, as provenance is a precondition for reproducibility. The code-fence rows are responses the API returned as successful, from which extraction yields the empty string: nine emitted a fence with nothing inside it beyond an optional language tag, and one wrote a pattern followed by an unmatched closing fence, which leaves the last fenced block empty. All ten are short replies of 2–18 completion tokens; none is a truncation against the 400-token cap.

Empty completions are classified by reported `finish_reason` into refusal, token-budget exhaustion, and unexplained, rather than being scored as empty patterns, which would be indistinguishable from a maximally poor answer.

B Operational hazards

We document configuration hazards encountered during this study, each of which produced plausible-looking but invalid data.

1. **Parameter combinations that are silently unroutable.** Setting `temperature` together with `require_parameters=true` yields a hard routing failure on 6 of 11 models. Our first pilot failed all 396 calls.

2. **Systems that cannot satisfy the experimental condition.** Two Gemini variants return *"Reasoning is mandatory for this endpoint"* and cannot be evaluated with reasoning disabled. Undetected, this would have placed one reasoning-enabled system in an otherwise reasoning-disabled comparison.

3. **Empty completions from budget exhaustion.** At a 200-token cap, reasoning models consumed the entire budget on hidden reasoning and returned no content.

4. **Resumption across configuration changes.** Our collection is resumable by (model, task, sample). An aborted run at one token cap left truncated rows that a resumed run at a different cap treated as complete, interleaving two configurations indistinguishably. Rows now carry a configuration fingerprint and resumption refuses on mismatch.

5. **Sample collapse in scoring.** An early scorer retained only the final sample per task, which would have rendered all @3 statistics disguised @1 statistics with no visible symptom.

6. **An estimator guard that is sound only on its own domain.** The PASS@k estimator is usually implemented with the shortcut "if $n - c < k$, return 1": if fewer than k samples failed, every choice of k must contain a success. That is correct for $n \geq k$, which is the only case the estimator is defined on, and it is silently wrong below it. A task that returned one sample satisfies $n - c \leq n < k$ whatever happened, so it scores a full 1.0 on every metric (pass_at_k$(1, 0, 3) = 1$) and a task nobody answered correctly is counted as answered correctly.

   We found this late, from a discrepancy of 1.3 points between two of our own tables, and it had been inflating exactly the two models that lost the most samples, which were the two at the top of the table. Correcting it moves `claude-opus-5` from second to fourth on USABLE@3 and makes `kimi-k3` rather than `claude-opus-5` the highest PASS@3. No claim in this paper turns on it (we publish no ranking, and Section 5.4 already says the band does not separate) but Table 1 is ordered by the affected column, and a reader comparing two tables would have had no way to tell which was right.

   The general form is a fast path justified by a precondition, in code that is also called when the precondition does not hold. The failure is silent, it is biased toward the systems with the most missing data, and nothing in the output looks wrong.

A content filter can silently bias a subset. Collecting StructuredRegex, Anthropic's content filter rejected 182 of 622 descriptions for `claude-opus-5`, all of them benign ("A string that starts with one or more digits and optionally ends with 'NU' or 'DG'."). A probe suggested the filter was stochastic: 8 of 20 rejected prompts succeeded when resent unchanged. Five retry rounds recovered 98, with the per-round yield collapsing (51, 31, 9, 2, 5), which shows a stochastic component over a hard core the filter rejects every time. That leaves 84 prompts the filter rejected on every attempt. A further 25 responses came back successful but yielded no pattern, and are not the filter's doing: 24 are a bare code fence of one or two tokens, the same failure mode as Section A's, and one is a genuine truncation, having spent the whole 400-token budget inside a `<thinking>` block. Together $84 + 25 = 109$, and final coverage was $622 - 109 = 513$.

The retry effort was not uniform across models, because nothing else needed it: only `claude-opus-5` was blocked, so only `claude-opus-5` received five extra rounds. The recovered 98 are therefore conditioned on non-refusal in a way the other ten models' answers are not. We do not think this biases regex quality (the filter keys on the description, which the model has not yet answered) but it is differential effort and the common-subset comparison below is the reason it does not matter for the reported numbers.

The recovery is not the interesting part. Scoring the other ten models on the 109 blocked tasks against the 513 kept shows the blocked tasks are 8.1 points harder on average (sd 3.3, negative for all ten). The surviving subset therefore *flattered* the affected model, at the same time as counting the blocked rows as failures *penalised* it. Either error alone is visible; together they produce a plausible number that means nothing. We report the common 513-task subset for all eleven models (Table 8) and release `runner/filter_bias_check.py`, which needs no API access.

C How the analysis was staged

The results in this paper were established in an order, and three of them replaced earlier readings of the same data. We record the sequence here because the checks that produced it are cheap, general, and easy to omit.

The composite came before its decomposition. We built USABLE as a conjunction of the corpus's three metrics, in the form the joint-benchmark literature reports (Section 2.1), and it showed a gap of about the size that literature would predict. Decomposing it (Section 4.3) attributes 87% of that gap to the equivalence term, and auditing that term (Section 5.1) shows it is dominated by reference error and underspecification. A composite of m terms is bounded above in reliability by its worst term, and reports a single number that hides which one is binding. The general lesson is that a joint metric should be published alongside its decomposition or not at all.

The human baseline came before its control. On this corpus alone, models screen safer than the human-written reference answers on 6 of eleven comparisons under an exact McNemar test with a Holm correction, which supports a claim about human practice at large. Extending the identical screen to five further populations (Section 4.5) does not support that claim, and replaces it with the execution ordering that is this paper's principal result. One population cannot distinguish a fact about people from a fact about the artifact that population happens to be. The cost of the extension was a few hours of screening and no inference at all.

Reference-independent did not mean corpus-independent. The 7.4% correctness-to-security penalty consults no answer key, which makes it more trustworthy than the composite and does not make it general. Re-running the same models and the same screen on StructuredRegex (Section 4.7) gives 16.5%. An effect size should be treated as corpus-local until a second corpus says otherwise, and a second corpus is usually cheaper than the first.

Two estimator defects were found by reading outputs, not summaries. The unbiased PASS@k estimator [Chen et al., 2021] returns 1.0 when $n - c < k$, which is correct for $n \geq k$ and scores a task with fewer than k returned samples as a full pass. Tasks with short samples are excluded from the @3 estimate here rather than credited (Section 3.4), and the fix moves one model by two places. Separately, the empirical timing pass initially reported every timeout as exponential; it now distinguishes growth order by the probe length at which the timeout first fired, and the degree split is discounted accordingly (Section 4.5.2). Both were visible in per-sample records and invisible in every aggregate we had computed.

D Reproduction

```
git clone https://github.com/plicara/regexeval-2026
cd regexeval-2026
make setup # pinned scorer + corpus download
make score RUN=sweep
```

Scoring reads only the committed generations. No API access, no expenditure. We verified reproduction from a clean checkout with fresh dependency installation and corpus download, obtaining bit-identical metrics. A reduced form of this check executes in continuous integration on every commit.