Repairing Regex Vulnerabilities via Localization-Guided Instructions

Sicheol Sung[*] **Joonghyuk Hahn**[*] **Yo-Sub Han**[†]
Department of Computer Science, Yonsei University, Seoul, Republic of Korea
{sicheol.sung,greghahn,emmous}@yonsei.ac.kr

Abstract

Regular expressions (regexes) are foundational to modern computing for critical tasks like input validation and data parsing, yet their ubiquity exposes systems to regular expression denial of service (ReDoS), a vulnerability requiring automated repair methods. Current approaches, however, are hampered by a trade-off. Symbolic, rule-based systems are precise but fail to repair unseen or complex vulnerability patterns. Conversely, large language models (LLMs) possess the necessary generalizability but are unreliable for tasks demanding strict syntactic and semantic correctness. We resolve this impasse by introducing a hybrid framework, localized regex repair (LRR), designed to harness LLM generalization while enforcing reliability. Our core insight is to decouple problem identification from the repair process. First, a deterministic, symbolic module localizes the precise vulnerable sub-pattern, creating a constrained and tractable problem space. Then, the LLM is invoked to generate a semantically equivalent fix for this isolated segment. This combined architecture successfully resolves complex repair cases intractable for rule-based repair while avoiding the semantic errors of LLM-only approaches. Our work provides a validated methodology for solving such problems in automated repair, improving the repair rate by 15.4%p over the state-of-the-art. Our code is available at https://github.com/cdltlehf/LRR.

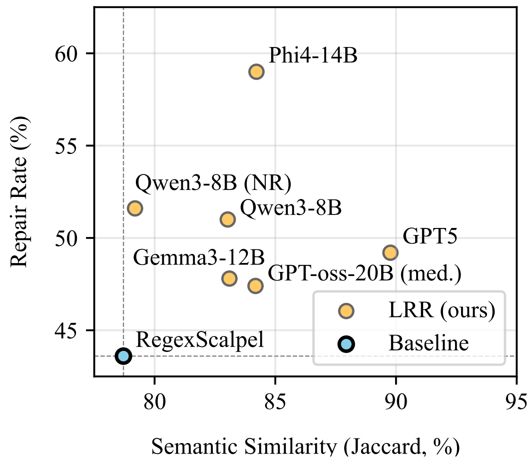


Figure 1: Performance comparison of the LRR framework against the baseline. The plot compares LRR implementations (yellow) against the rule-based baseline (blue), demonstrating successful improvement in repair rate (y-axis) across semantic similarity (x-axis).

1 Introduction

Regular expressions (regexes) are a foundational tool in modern software development, serving as the backbone for critical tasks such as input validation, data parsing, and network security policies (Chapman and Stolee, 2016; Cortes and Mohri, 2008; Liu et al., 2012). Their concise syntax and powerful pattern-matching capabilities have made them indispensable. However, the expressive power of modern regexes introduces a critical performance vulnerability: certain regexes can exhibit super-linear, often exponential, time complexity. A ReDoS attack exploits this vulnerability (Rathnayake and Thielecke, 2014; Turonová et al., 2022). In such an attack, a malicious user submits a crafted string that forces the regex engine into a prolonged, resource-intensive matching process, leading to resource exhaustion and service failure. The severity of this threat is evidenced by high-profile incidents at companies like StackOverflow and Cloudflare, while numerous other systems remain potentially vulnerable (Bhuiyan et al., 2024).

The primary strategy for mitigating ReDoS vulnerabilities is the automated repair of vulnerable regexes themselves. An alternative, replacing the underlying regex engine, is often impractical due to high engineering costs and the risk of introducing subtle semantic changes in matching behavior (Davis et al., 2019). However, traditional repair

[*]Equal contribution.
[†]Corresponding author.

Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics
Volume 1: Long Papers, pages 8128–8142

approaches have notable limitations. Synthesis-based methods require a large set of examples to guide the repair process (Li et al., 2020). In contrast, rule-based systems rely on human-designed heuristics that offer precision and are effective at localizing known vulnerability patterns. However, they are fundamentally brittle; they often fail to generate semantically-equivalent and generalizable repairs for complex vulnerability patterns.

We propose the localized regex repair (LRR) framework, a hybrid framework designed to integrate the strengths of both approaches. The framework combines the precision of symbolic analysis, which can accurately localize the vulnerable part of a regex, with the generalization capabilities of LLMs (Wei et al., 2022a; Chowdhery et al., 2023; Zan et al., 2023). By pinpointing the exact source of the vulnerability, we can provide the LLM with a precise context, allowing it to generate a targeted and semantically correct repair. This localization-guided approach steers the powerful generation abilities of LLMs (Geng et al., 2024; Hirsch et al., 2025; Orvalho et al., 2025). Figure 1 summarizes the increased repair rate and preserved semantic similarity achieved by our method.

2 Related works

In this section, we review existing research related to ReDoS, including methods for vulnerability detection and mitigation, as well as approaches that utilize symbolic analysis.

2.1 Localization of large problems

Recent works have shown that tackling code-related tasks by narrowing focus to local text can improve both efficiency and accuracy (Zhang et al., 2022; Prenner and Robbes, 2024). For instance, Tang et al. (2025) demonstrate that program repair benefits from a staged approach: the model first attempts fixes using only the immediate local code snippet, expanding to dependency-related context only when necessary. Similarly, Pamies-Juarez et al. (2013) provides evidence that code repair can become more efficient by relying on small, localized subsets rather than global redundancy.

In the broader LLM field, context pruning techniques (Li et al., 2023; Huang et al., 2024) validate this strategy by showing that selectively removing redundant portions of long documents can reduce computational cost while maintaining performance. This established efficacy of localization in code repair strongly suggests that precisely isolating the vulnerable sub-pattern enables robust and efficient LLM-based regex vulnerability repair.

2.2 In-context learning

In-context learning (ICL) is a core capability of LLMs to learn new tasks from a few examples provided directly in the prompt, without any weight updates (Brown et al., 2020; Min et al., 2022b; Shivagunde et al., 2024). While early ICL was limited to simple input-output pairs, chain-of-thought (CoT) prompting was introduced as a sophisticated advancement to handle more complex reasoning (Min et al., 2022a; Wei et al., 2022b). CoT provides an intermediate, step-by-step reasoning process to arrive at a solution and guides the LLM to emulate a similar deductive procedure. CoT has significantly improved performance on intricate analytical tasks (Zhou et al., 2023; Sun et al., 2024). The success of CoT has inspired more advanced variants of CoT (Wang et al., 2023b), such as graph-of-thoughts (Yao et al., 2024) or tree-of-thoughts (Ranaldi et al., 2024) to explore reasoning paths more effectively. Following the essence of such works, we design a CoT instruction specifically designed to repair ReDoS.

2.3 Vulnerability detection

Vulnerability detection efforts include identifying common ReDoS anti-patterns to guide developers during regex writing (Hassan et al., 2023). Dynamic analysis tools like ReDoSHunter (Li et al., 2021) classify vulnerabilities using signature-based methods for attack string generation. Further research employs fuzzing and advanced dynamic methods to accelerate problematic string generation and improve detection accuracy (McLaughlin et al., 2022; Wang et al., 2023a; Su et al., 2024).

Studies by Siddiq et al. (2024a) and Siddiq et al. (2024b) show that LLMs struggle to inherently generate ReDoS-invulnerable regexes, often producing vulnerabilities similar to those found in human code. Our approach addresses this limitation by using a precise, rule-based module to localize the vulnerability, transforming the problem into a manageable repair task where generative strengths of the LLM can be effectively applied.

2.4 Vulnerability repair

FlashRegex (Li et al., 2020) uses programming-by-example approach to compose ReDoS-invulnerable regexes from examples. They guarantee the invulnerability by enforcing 1-unambiguity during synthesis. Remedy (Chida and Terauchi, 2022) supports symbolic regexes which allow interval character-classes instead of literals, and extended features such as look-aheads and back-references. RegexScalpel (Li et al., 2022) identifies and repairs only the vulnerable spans of a given regex, minimizing semantic deviations while applying example-based synthesis. While these synthesis-based methods are powerful, they are often rigid and require a large set of examples to guide the repair process. Our approach, in contrast, leverages the flexibility of LLMs to generate nuanced and correct repairs without needing an extensive, manually-curated example.

3 Problem definition

The core problem we address is the automated repair of regex that are vulnerable to ReDoS attacks. These vulnerabilities stem from catastrophic backtracking, where certain regexes exhibit super-linear time complexity on specific inputs. While the prevalence and danger of these vulnerabilities are well-established, effective and scalable repair remains a significant challenge.

Traditional repair methods face critical limitations: synthesis-based approaches require large, often unavailable, example sets, while rule-based systems are inflexible and fail to generalize despite being precise for known patterns.

Given a vulnerable regex s, our objective is to synthesize a new repaired regex t that is safe from ReDoS attacks and semantically equivalent to s. Our work operates on the hypothesis that a naive approach using an LLM is ineffective for this task due to its difficulty in understanding the complex and subtle causes of a vulnerability. We overcome this limitation by proposing a hybrid framework that leverages a symbolic analysis module to precisely localize the vulnerable sub-pattern. This process transforms a broad, complex problem into a smaller, more manageable one, allowing the LLM to focus on generating a targeted and correct repair.

Problem. *Given a ReDoS vulnerable regex s, our goal is to compose a repaired regex t that is:*

1. *ReDoS-invulnerable, and*
2. *semantically and syntactically similar to s.*

4 Method

Our approach addresses the core limitation of naive LLM-based repair: their poor performance on complex regular expressions due to a lack of precise vulnerability analysis. We adopt a symbolic-guided in-context learning methodology, inspired by the localize-and-fix strategy, to enable LLMs to perform targeted and effective repairs. The LRR framework leverages the complementary strengths of symbolic analysis and LLMs. The symbolic module serves as a precise localization engine, identifying the exact sub-pattern responsible for the ReDoS vulnerability. We integrate this output into a structured prompt for the LLM. This focused context guides the LLM to concentrate its generative power on the specific area of concern. The task of the LLM is to leverage this focused context to generate a semantically equivalent, but ReDoS-invulnerable, repaired version of the original vulnerable regex.

The symbolic module in our framework functions as a precise vulnerability localizer, directly leveraging the established heuristics from existing rule-based ReDoS repair methods. This approach is based on previous works showing that while rule-based methods are highly effective for merely detecting ReDoS vulnerability (Hassan et al., 2023; Li et al., 2021), they often fail to repair the detected vulnerability. Specifically, the module uses an analyzer to pinpoint the vulnerable sub-pattern s' within the original regex s corresponding to a known ReDoS pattern.

We then teach the LLM how to utilize this localization information by providing few-shot chain-of-thought (CoT) examples. Each CoT example consists of the vulnerable regex, the reasoning process and the repaired regex. The reasoning details how the given sub-pattern s' causes the ReDoS vulnerability and how rule-based repair methods modify it to resolve the issue. Through this approach, we aim for the LLM to not only learn the given heuristics but also to leverage this prior knowledge to identify vulnerable sub-patterns missed by existing heuristics or to apply repair independently of the heuristic for the given sub-pattern.

5 Experimental setup

We define the specific settings and evaluation criteria for the evaluation. We use: Qwen3 (Yang et al., 2025), Llama3.1 (Weerawardhena et al., 2025), Gemma3 (Kamath et al., 2025), Phi4 (Abdin et al.,

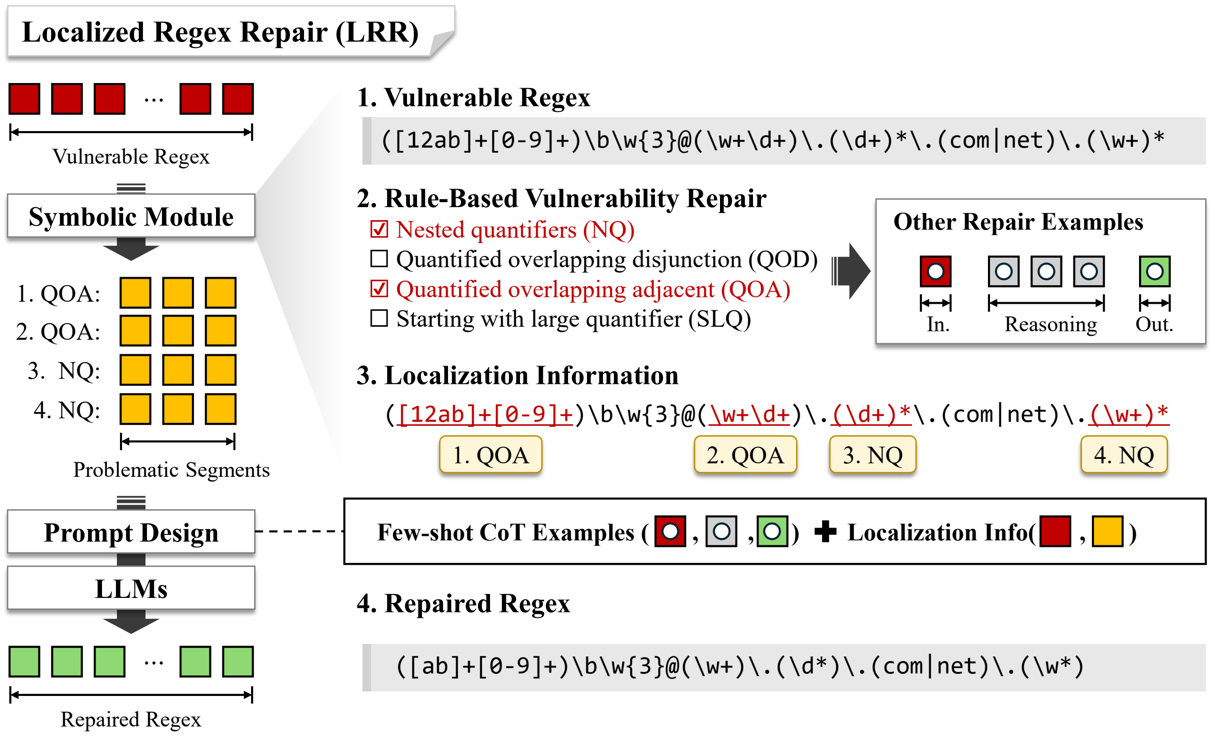


Figure 2: An overview of our LRR framework. Given a vulnerable regex, the symbolic module first analyzes and localizes the vulnerability-inducing sub-pattern. This identified sub-pattern is then integrated into the LLMs' prompt, allowing the models to focus on the problematic segment and maximize the repair rate.

2024), DeepSeek-R1-Distill-Qwen (DeepSeek-AI et al., 2025), GPT-oss (Agarwal et al., 2025), and GPT5 (OpenAI, 2025). We outline the specific software and hardware specification in Appendix A.

5.1 Prompt design

We design our prompts by analyzing the successful repair processes of RegexScalpel, a rule-based regex repair tool. The heuristics used by Scalpel are categorized into four major vulnerability types: nested quantifiers, quantified overlapping disjunction (QOD), quantified overlapping adjacent and starting with large quantifier. We convert these heuristics into a natural language chain-of-thought reasoning process to effectively guide LLMs.

For the 5-shot prompt, we select a single example that incorporates three of the four major repair rule categories, excluding the QOD type. Since no single example showcases all four rule types at once, we intentionally choose a sample that demonstrates a multi-step repair. We then add four more examples, each showing a distinct rule category applied in isolation. This exposes the model to a wider variety of vulnerability causes and teaches it that a regex repair might involve multiple, successive steps. This method aids the model in learning how to apply different rules and recognize that the repair process as a combination of several logical steps depending on the complexity of vulnerability. The prompts are detailed in Appendix B.

5.2 Metric

The main purpose of the repair is to eliminate ReDoS vulnerabilities while minimally transforming the original regex. Changes introduced during repair process are either syntactic or semantic. First, syntactic alterations can lead to complex regexes, which can harm readability and reusability (Davis et al., 2019). On the other hand, semantic changes can defeat the original purpose of the regex. Therefore, an ideal repair must not only remove the vulnerability but also maintain syntactic and semantic similarity to the original regex.

5.2.1 Repair rate

We define the *repair rate* as the proportion of vulnerable regexes that are successfully repaired. We treat ill-formed resulting regexes or zero Jaccard similarity with the original regexes as failure cases; see Section 5.2.3 for details on Jaccard similarity. We include vulnerability elimination rate in Section C, which we exclude in the main text since it serves as an insufficient measure of success

Note that there is no consensus on a formal definition of ReDoS-invulnerability and it depends on engine implementation (Sung et al., 2022; Bhuiyan et al., 2024). We adopted ReDoSHunter (Li et al., 2021) as it is the current state-of-the-art dynamic detector, ensuring our evaluation aligns with standard practices in conventional studies. It handles practical regex features such as look-arounds, back-references, and anchors more reliably than many static analysis tools. Unlike some other dynamic methods, it consistently produces a result within a reasonable time. For our evaluation, we classify any regex as invulnerable if it cannot identify a vulnerability within one-minute timeout threshold.

5.2.2 Syntactic similarity

Prior work on generating theoretically safe regexes, such as NFA-to-DFA conversion, often yields patterns of doubly exponential size (van der Merwe et al., 2017). While semantically robust, the resulting patterns are practically unusable, as developers already find even standard regexes challenging to read and maintain (IV et al., 2019). Therefore, the syntactic score serves as a vital metric for developer adoption, ensuring that the repaired pattern maintains a familiar structure that facilitates future reusability and maintenance.

We quantify syntactic similarity between an original regex s and a repaired regex t using two metrics. First, the relative length increase (RLI) measures the preservation of the regex's size:

$$\mathrm{RLI}(s,t) = \frac{|t| - |s|}{|s|}.$$

An RLI value close to zero indicates that the regex's length has been maintained. We do not define the RLI for ill-formed regexes, as there is no meaningful reference point (worst-case value).

The normalized Levenshtein similarity (NLS) measures the number of character-level structural changes:

$$\mathrm{NLS}(s,t) = 1 - \frac{\mathrm{LD}(s,t)}{\max(|s|,|t|)},$$

where $\mathrm{LD}(s,t)$ is the Levenshtein distance, representing the minimum number of single-character insertions, deletions or substitutions required to change s into t. We give zero NLS scores for ill-formed results.

5.2.3 Semantic similarity

Determining the equivalence of regexes is PSPACE-complete (Stockmeyer and Meyer, 1973) and even undecidable for modern features (Freydenberger, 2013). Thus, we approximate the semantic similarity using sampling. For each regex s, we first generate a sample language $\hat{L}(s)$ to measure the semantic similarity between regexes. For each regex, we create a sample language of size 100 using `xeger` package,[1] which performs a random walk on the regex and generates a sample string matched by the regex. We allow duplicates to ensure that regexes with sparse languages (matching few strings) and those with dense languages (matching many) are weighted equally in our analysis.

Given an original regex s and a repaired regex t, we evaluate their semantic similarity by counting true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). These metrics are calculated over the union $\hat{L}(s) \cup \hat{L}(t)$ of their samples. A string is considered a positive instance if the original regex s matches it.

Using these counts, we can define standard evaluation metrics such as precision and recall:

$$\text{Precision} = \frac{\text{TP}}{\text{TP}+\text{FP}}, \quad \text{Recall} = \frac{\text{TP}}{\text{TP}+\text{FN}}.$$

High precision corresponds to soundness; it indicates that strings matched by the repaired regex t are also likely to be matched by the original regex s. Conversely, high recall corresponds to completeness; it suggests that strings matched by the original regex s are also likely to be captured by the repaired regex t.

The Jaccard similarity provides a unified view of these metrics, measuring the overlap between the two regexes:

$$\text{Jaccard similarity} = \frac{\text{TP}}{\text{TP}+\text{FP}+\text{FN}}.$$

A high Jaccard similarity indicates a strong overall agreement between the original and repaired regexes, where both high precision and high recall are required. Also, we give zero precision, recall and Jaccard similarity scores for ill-formed results as well as NLS.

We opt for Jaccard similarity over the F1 score because it better reflects the set-based nature of languages, directly measuring the overlap between two sets of strings. Since Jaccard similarity and the F1 score are monotonically related, they preserve the same ranking, ensuring our choice does not alter the relative performance assessment.

[1] https://pypi.org/project/xeger/

Method	Model	W.F.	Repair	Syntactic sim.		Semantic similarity		
				RLI (↓)	NLS	Jaccard	Prec.	Recall
Baseline	RegexScalpel	**100.00**	43.60	100.53	69.60	78.71	**99.91**	78.77
	Remedy	46.10	26.00	102.94	21.23	12.66	37.30	13.64
LRR (ours)	Qwen3-8B (NR)	98.60	51.60	79.75	69.93	79.19	92.53	80.83
	Llama3.1-8B	96.80	50.00	119.97	58.52	73.06	85.08	74.41
	Gemma3-12B	99.40	47.80	63.27	**76.68**	83.10	93.27	84.87
	Phi4-14B	99.30	**59.00**	100.43	63.37	84.22	95.78	85.93
	Qwen3-8B	97.90	51.00	68.79	73.33	83.03	94.47	84.23
	DS-R1-Qwen-7B	80.70	45.60	72.44	59.34	63.86	75.03	64.86
	GPT-oss-20B (low)	91.00	47.50	63.02	69.41	78.89	88.60	79.64
	GPT-oss-20B (med.)	96.80	47.40	**61.11**	74.45	84.18	94.68	84.82
	GPT5	99.70	49.20	62.91	75.57	**89.77**	98.96	**90.24**

Table 1: A performance evaluation of two baseline methods (RegexScalpel and Remedy) against our LRR framework using various LLMs. Qwen3-8B (NR) represents the model operating in the non-reasoning mode. The W.F. column denotes the ratio of well-formed output regexes that are compilable. All scores are denoted in percent (%) and best scores are marked in bold for each metric.

This entire semantic analysis is conducted using Python's `Lib/re` module. While some prior work, like RegexScalpel, was evaluated against a Java matcher, which has only minor differences between them (Davis et al., 2019).

5.3 Dataset

Our evaluation dataset comes from the Polyglot corpus (Davis et al., 2019), whose divergency ensures our findings are generalizable and not biased toward a specific language field. We first identify regexes vulnerable to ReDoS attacks using the ReDoSHunter (Li et al., 2021) from the Polyglot corpus. We then filter these results to include only regexes whose length is less than 1,024 to meet our context size requirements. From this filtered set, we constructed our dataset of 1,000 regexes to analyze the performance regex repair tools.

We select a balanced sample consisting of two distinct groups: 500 regexes that result in invulnerable patterns and 500 regexes where the tool fails to complete within one minute or where the resulting regexes remain vulnerable. We note that the first group, despite yielding invulnerable results, does not guarantee a successful repair since it includes cases with zero Jaccard similarity. We provide details in Appendix D.1.

This balanced dataset allows us to not only measure our method's ability to fix difficult, unresolved regexes but also to understand the structural and semantic changes that occur during the repair process. By analyzing both successful and failed repairs, we evaluate LRR's performance and demonstrate its capability to overcome the limitations of existing rule-based approaches.

6 Experimental results and analysis

The overall results of our performance evaluation are presented in Table 1; the full version is provided in Appendix F. The table shows the performance increase obtained by LRR, which incorporates both RegexScalpel and LLM variants. Among the LRR variants, the top four rows represent results of non-reasoning models, while the bottom five rows represent reasoning models.

6.1 Baseline comparisons

We first examine the primary baseline, RegexScalpel. The method shows a perfect score for well-formed ratio but is limited by a repair rate 43.60%, failing to resolve a majority of the vulnerabilities. We present such examples that are then resolved by LRR in Section 6.6. RegexScalpel often results in longer regexes, increasing the regex length by approximately a factor of two. We include analysis on the secondary baseline, Remedy, due to its low well-formed ratio, which makes it unsuitable for real-world usage; we discuss this impracticality in Appendix D.2.

In contrast, LRR surpasses the repair rates by 6.30%p on average, increasing the rate from 43.60% to 49.90%. Most notably, LRR with Phi4 demonstrates the best repair rate 59.00%, achieving a 15.40%p increase over RegexScalpel. This highlights the effectiveness of combining symbolic analysis with generalizability of LLMs.

A trade-off exists regarding reasoning: the non-reasoning LLM, Phi4, improves the repair rate by 8%p over the reasoning counterpart, Qwen3, from 51.00% to 59.00%. However, this gain is accompanied by a 31.64%p increase in RLI from 68.79% to 100.43%. This empirically proves that vulnerability repair often requires an input regex to be much longer; we detail this analysis in Section 6.4.

6.2 Comparison with pure LLM baseline

We compared our framework against a pure LLM baseline using Phi-4. In this setting, the model utilizes only the basic prompt (detailed in Appendix B) to repair the entire regex without any symbolic localization aid from RegexScalpel.

Method	Repair	NLS	Jaccard
LRR (Phi-4)	**59.00**	63.37	**84.22**
Pure LLM (Phi-4)	32.90	**82.44**	82.94

Table 2: Performance comparison between LRR and Pure LLM baseline.

The results demonstrate that the pure LLM baseline significantly underperforms the LRR framework (32.9% vs. 59.0% repair rate). This confirms that symbolic localization is essential for guiding the LLM to the specific fault.

6.3 Ablation studies

We conduct an ablation study of LRR to evaluate the individual contributions of our prompt components: vulnerability localization and 5-shot CoT prompting. The analysis in Table 3 uses Phi4 and Qwen3 as representatives of the best-performing non-reasoning and reasoning LLMs, respectively.

The results reveal a distinct dependency for model types. For the non-reasoning Phi4 model, the 5-shot CoT prompting is the most critical component. This removal causes a significant drop in the repair rate, suggesting that the models rely on the examples to learn the procedural knowledge of the repair task. In contrast, the reasoning-based Qwen3 model is most impacted by removing localization. This indicates that it requires precise guidance on where to apply its reasoning.

Consequently, the study confirms that the two prompt components are synergistic and complementary. Localization tells the model what to fix, which is especially crucial for reasoning models, while the 5-shot CoT prompting teaches the models how to fix, specifically effective for guiding non-reasoning models. By providing both a focused problem scope and a clear solution procedure, LRR is robustly effective across different LLMs.

Model	Repair	NLS	Jaccard
RegexScalpel	43.60	30.40	78.71
Phi4-14B	59.00	63.37	84.22
(-) localization	57.80	65.72	83.61
(-) 5-shot	49.70	72.40	76.91
(-) local. & 5-shot	49.00	26.95	78.88
Qwen3-8B	51.00	73.33	83.03
(-) localization	46.60	77.10	83.61
(-) 5-shot	47.60	75.39	79.11
(-) local. & 5-shot	47.30	75.74	79.58

Table 3: Ablation study on our prompt components: vulnerability localization and 5-shot CoT prompting.

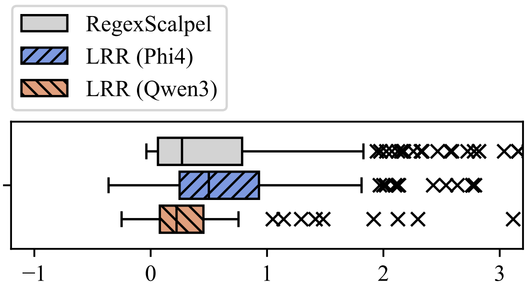
6.4 Analysis on syntactic similarity

Figure 3 illustrates the distribution of the two syntactic similarity scores; each box represents the inter-quartile range (IQR), with the whiskers extending up to 1.5 times the IQR. Both RegexScalpel and Phi4 exhibit a higher RLI compared to the Qwen3 variant.

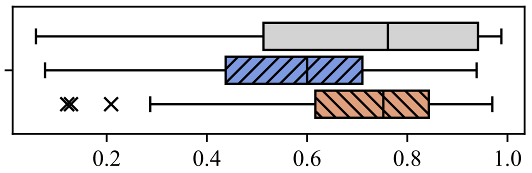
This disparity suggests that RegexScalpel and Phi4 have a higher tendency to add constructs like look-aheads during the repair process, whereas the Qwen3 model shows a relatively lower propensity for such additions. Also, the lower propensity implies that the Qwen3 model performs repair in a relatively more stable manner.

The stability of the reasoning model stems from a tendency to apply the simplest strategy with the minimal modifications first, and then it stops when the vulnerability is resolved. On the other hand, the non-reasoning models often apply all applicable rules at once. Considering that most ReDoS vulnerability resolution strategies involve increasing the regex length, the unnecessary application of additional strategies contributed to the reduced syntactic similarity of the non-reasoning model.

We provide detailed examples illustrating these behavioral differences between the reasoning and non-reasoning models in Appendix E.

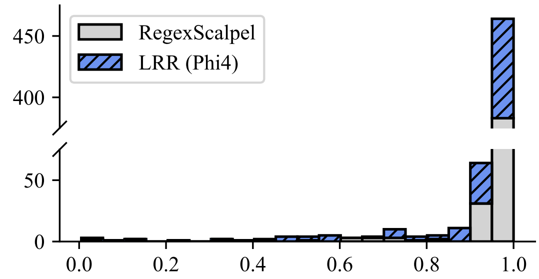


(a) Relative length increase

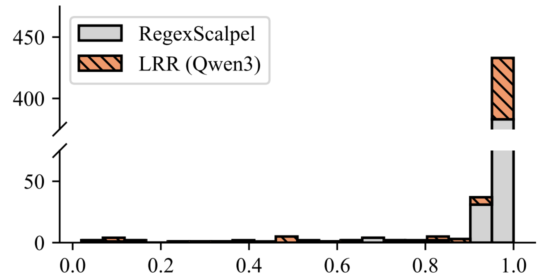


(b) Normalized Levenshtein similarity

Figure 3: Box-and-whisker plots of the RLI (top) and NLS (bottom) of successfully repaired cases for each method. For the LRR method, we exclude all cases successfully repaired by the baseline RegexScalpel to isolate and highlight the unique contribution of LLMs.



(a) LRR with non-reasoning LLM; Phi4-14B



(b) LRR with reasoning LLM; Qwen3-8B

Figure 4: Histograms illustrating the distribution of Jaccard Similarity scores for successfully repaired regexes.

The distribution of NLS shows an inverse trend compared to the RLI scores. Given that an RLI close to zero indicates the repaired regex maintains its original length, this inverse relationship suggests that repair is primarily achieved by adding characters to the regex.

6.5 Analysis on semantic similarity

The primary goal of LRR is to produce a robust and semantically equivalent repair. Figure 4 illustrates the distribution of successful repair scores and Jaccard similarity for LRR with Phi4 and Qwen3 as strong non-reasoning and reasoning LLMs, respectively. The analysis reveals a trade-off between the quantity of repairs and their semantic correctness.

LRR with Phi4 successfully captures and repairs vulnerable regexes that RegexScalpel fails, shown in Figure 4a. However, the distribution of these similarity scores, while strong, is less concentrated at the full-score (1.0) mark than its reasoning-based counterpart. Figure 4b illustrates that Qwen3 produces repairs of more reliable semantic quality. Although Phi4 shows the most effective repair rate, a significantly large proportion of successful repairs by Qwen3 achieve near-perfect Jaccard similarity. This suggests that the reasoning model is more capable of maintaining semantic correctness while repairing vulnerabilities.

Perhaps, its high precision (99.91%) is consistent with the claim by Li et al. (2022) that their heuristic is designed to never expand the pattern's languages, thereby preventing false positives of the repaired pattern. In contrast, the LRR methodology exhibits a slightly reduced precision, by aiming to eliminate a greater number of vulnerabilities.

6.6 Case-study

We analyze cases where LRR succeeds while symbolic baselines such as RegexScalpel and Remedy fail to reveal why our approach improves the overall repair rate.

The following examples highlight the key advantages of LLMs, specifically their ability to perform contextual reasoning. This allows the LLMs to infer the semantic intent of given regexes—what the expression is meant to match—rather than restricting the analysis solely on the syntactic structure. This context-aware understanding enables LRR to recognize and repair the regexes that pure rule-based approaches cannot address.

Examples: contextual repairs

Original:
```
<\?(=|php)(.+?)\?>
```
Repaired:
```
<\?(=|php)([^\?>]+?)\?>
```

```
Original:
  <a href="([^"]+)">(.+?)</a>
Repaired:
  <a href="([^"]+)">([^<]+)</a>
```

```
Original:
  @@ \-(\d+),?(\d+)? \+(\d+),?(\d+)? @@
Repaired:
  @@ \-(\d{1,3}(?:,\d{2})?) \+(\d{1,3}... @@
```

This capability to understand the context of regexes is especially evident in domain-specific formats. For instance from the above example, the LLM recognizes the `@@...@@` signature as a git diff header and infers that `\d+` represents line counts. Consequently, the LLMs constrains the pattern to a more plausible range. This demonstrates the effectiveness of contextual understanding of LLMs in our LRR framework that moves beyond a rule-based approach.

6.7 Failure analysis

Despite these successes in leveraging contextual understanding, the LRR framework is not without limitations. We conducted a detailed analysis of failure cases for Phi4 and identified two primary types of repair failure.

Failure to remove vulnerability. The model occasionally fails to fully eliminate the vulnerability when the applied repair heuristic proves insufficient. In such cases, the model merely mimics surface-level heuristics of RegexScalpel without achieving the necessary generalization. As shown in the example below, while the model successfully bounds the first quantifier, it neglects the second group, leaving the regex vulnerable.

Example: failure to remove vulnerability

```
Original:
  ^(.*)end,running hash(.*),end$
Repaired:
  ^(.{0,1000})end,running hash(.*),end$
```

Failure to preserve semantics. Semantic failures—often resulting in near-zero Jaccard similarity—stem primarily from format hallucinations or incorrect heuristic applications. The former arises from output generation errors, such as excessive escaping, rather than a fundamental lack of regex comprehension; for instance, a valid regex can be broken by double-escaped backslashes. The latter occurs when the model applies strategies absent from the few-shot examples. A critical instance is the misinterpretation of negative look-ahead (`(?!...)`) demonstrations as semantically distinct positive look-aheads (`(?=...)`), a behavior likely driven by the model's training bias toward the more prevalent positive look-ahead patterns found in general corpora.

Examples: failure to preserve semantics

```
Original:
  inet[^\d]+([\d\.]+)\s
Repaired:
  inet[^\\d]{0,100}([\\d\\.]+)\\s
```

```
Original:
  <body>([\w\W]+)@RENDER_ENTRY@
Repaired:
  <body>(.*?)(?=@RENDER_ENTRY@)
```

7 Conclusion

We propose and validate a hybrid framework that addresses the core challenge of automated ReDoS repair. Traditional methods are fundamentally limited: synthesis-based approaches require large data dependency, while rule-based systems lack the generalizability.

Our framework overcomes these limitations by integrating the localize-and-fix strategy with the advanced generative capabilities of LLMs. The symbolic module performs precise vulnerability localization, identifying the vulnerable sub-pattern. The LLM leverages this focused, actionable context to perform the repair, effectively transforming the broad, complex task into a manageable one.

Experiments confirm this approach successfully repairs complex patterns that traditional rule-based systems fail to address. The resulting regexes not only eliminate the vulnerability but also show high syntactic and semantic similarity, ensuring the patterns remain both safe and practical.

In conclusion, our work demonstrates a robust and effective method for automated ReDoS vulnerability repair, offering a promising path for developing more adaptable and scalable solutions by combining precise analytical tools with powerful generative models.

Limitations

The LRR framework operates under two primary, empirical constraints. First, the symbolic localization module relies on established heuristics derived from existing rule-based repair methods. This dependence means the effectiveness of LLMs is constrained by the knowledge of the underlying analyzer, potentially limiting its ability to precisely localize vulnerability of unseen patterns. If the localization module fails to pinpoint the vulnerable sub-segment, the LLM is forced to analyze the entire, unconstrained regex, which reduces the efficiency benefit of the localize-and-fix strategy.

Second, the validation of our repair success lacks formal safety guarantees as it relies on the dynamic detection tool ReDoSHunter. Moreover, given that the definition of a ReDoS vulnerability often depends on context and engines, there is no single, universally consistent standard for definitive invulnerability (Hassan et al., 2023). Thus, the reported repair rate is best viewed as robust evidence of practical safety relative to the fidelity of the specific detection mechanism employed.

Acknowledgments

This research was supported by the KISA grant (RS-2025-02222626) and the AI Graduate School Program (RS-2020-II201361) funded by the Korean government.

References

Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J Hewett, Mojan Javaheripi, Piero Kauffmann, and 1 others. 2024. Phi-4 technical report. *arXiv preprint arXiv:2412.08905*.

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K. Arora, Yu Bai, Bowen Baker, Haiming Bao, Boaz Barak, Ally Bennett, Tyler Bertao, Nivedita Brett, Eugene Brevdo, Greg Brockman, Sébastien Bubeck, Che Chang, Kai Chen, and 105 others. 2025. gpt-oss-120b & gpt-oss-20b model card.

Masudul Hasan Masud Bhuiyan, Berk Çakar, Ethan H Burmane, James C Davis, and Cristian-Alexandru Staicu. 2024. SoK: A literature and engineering review of regular expression denial of service. *arXiv preprint arXiv:2406.11618*.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems, NeurIPS*.

Carl Chapman and Kathryn T. Stolee. 2016. Exploring regular expression usage and context in Python. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*, pages 282–293.

Nariyoshi Chida and Tachio Terauchi. 2022. Repairing DoS vulnerability of real-world regexes. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 2060–2077.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, and 48 others. 2023. PaLM: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24:240:1–240:113.

Corinna Cortes and Mehryar Mohri. 2008. Learning with weighted transducers. In *Proceedings of the 7th International Workshop on Finite State Methods and Natural Language Processing*, pages 14–22.

James C. Davis, Louis G. Michael IV, Christy A. Coghlan, Francisco Servant, and Dongyoon Lee. 2019. Why aren't regular expressions a lingua franca? An empirical study on the re-use and portability of regular expressions. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*, pages 443–454. ACM.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 81 others. 2025. DeepSeek-R1: Incentivizing reasoning capability in llms via reinforcement learning.

Dominik D Freydenberger. 2013. Extended regular expressions: Succinctness and decidability. *Theory of Computing Systems*, 53(2):159–193.

Saibo Geng, Berkay Döner, Chris Wendler, Martin Josifoski, and Robert West. 2024. Sketch-guided constrained decoding for boosting blackbox large language models without logit access. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics, ACL*, pages 234–245.

Sk Adnan Hassan, Zainab Aamir, Dongyoon Lee, James C. Davis, and Francisco Servant. 2023. Improving developers' understanding of regex denial of service tools through anti-patterns and fix strategies. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1238–1255.

Eran Hirsch, Aviv Slobodkin, David Wan, Elias Stengel-Eskin, Mohit Bansal, and Ido Dagan. 2025. LAQuer: Localized attribution queries in content-grounded generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 15355–15370.

Xijie Huang, Li Lyna Zhang, Kwang-Ting Cheng, Fan Yang, and Mao Yang. 2024. Fewer is more: Boosting math reasoning with reinforced context pruning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 13674–13695.

Louis G. Michael IV, James Donohue, James C. Davis, Dongyoon Lee, and Francisco Servant. 2019. Regexes are hard: Decision-making, difficulties, and risks in programming regular expressions. In *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*, pages 415–426. IEEE.

Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean-Bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, Gaël Liu, and 191 others. 2025. Gemma 3 technical report.

Yeting Li, Zixuan Chen, Jialun Cao, Zhiwu Xu, Qiancheng Peng, Haiming Chen, Liyuan Chen, and Shing-Chi Cheung. 2021. ReDoSHunter: A combined static and dynamic approach for regular expression DoS detection. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 3847–3864.

Yeting Li, Yecheng Sun, Zhiwu Xu, Jialun Cao, Yuekang Li, Rongchen Li, Haiming Chen, Shing-Chi Cheung, Yang Liu, and Yang Xiao. 2022. RegexScalpel: Regular expression denial of service (ReDoS) defense by Localize-and-Fix. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 4183–4200, Boston, MA. USENIX Association.

Yeting Li, Zhiwu Xu, Jialun Cao, Haiming Chen, Tingjian Ge, Shing-Chi Cheung, and Haoren Zhao. 2020. Flashregex: Deducing anti-ReDoS regexes from examples. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pages 659–671.

Yucheng Li, Bo Dong, Frank Guerin, and Chenghua Lin. 2023. Compressing context to enhance inference efficiency of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 6342–6353.

Tingwen Liu, Yong Sun, Alex X. Liu, Li Guo, and Binxing Fang. 2012. A prefiltering approach to regular expression matching for network security systems. In *Proceedings of the 10th International Conference on Applied Cryptography and Network Security*, pages 363–380.

Robert McLaughlin, Fabio Pagani, Noah Spahn, Christopher Kruegel, and Giovanni Vigna. 2022. Regulator: Dynamic analysis to detect ReDoS. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 4219–4235.

Sewon Min, Mike Lewis, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2022a. MetaICL: Learning to learn in context. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL*, pages 2791–2809.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2022b. Rethinking the role of demonstrations: What makes in-context learning work? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 11048–11064.

OpenAI. 2025. GPT-5 System Card.

Pedro Orvalho, Mikolás Janota, and Vasco M. Manquinho. 2025. Counterexample guided program repair using zero-shot learning and MaxSAT-based fault localization. In *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence*, pages 649–657.

Lluis Pamies-Juarez, Henk D. L. Hollmann, and Frédérique E. Oggier. 2013. Locally repairable codes with multiple repair alternatives. In *Proceedings of the 2013 IEEE International Symposium on Information Theory*, pages 892–896.

Julian Aron Prenner and Romain Robbes. 2024. Out of context: How important is local context in neural program repair? In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE*, pages 83:1–83:13.

Leonardo Ranaldi, Giulia Pucci, Federico Ranaldi, Elena Sofia Ruzzetti, and Fabio Massimo Zanzotto. 2024. A tree-of-thoughts to broaden multi-step reasoning across languages. In *Findings of the Association for Computational Linguistics: NAACL*, pages 1229–1241.

Asiri Rathnayake and Hayo Thielecke. 2014. Static analysis for regular expression exponential runtime via substructural logics. *arXiv preprint CoRR*, 1405.7058.

Namrata Shivagunde, Vladislav Lialin, Sherin Muckatira, and Anna Rumshisky. 2024. Deconstructing in-context learning: Understanding prompts via corruption. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation, LREC/COLING*, pages 4509–4529.

Mohammed Latif Siddiq, Jiahao Zhang, Lindsay Roney, and Joanna C. S. Santos. 2024a. 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'24, page 52–56. Association for Computing Machinery.

Mohammed Latif Siddiq, Jiahao Zhang, and Joanna Cecilia Da Silva Santos. 2024b. 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 '24, page 190–201. Association for Computing Machinery.

L. J. Stockmeyer and A. R. Meyer. 1973. Word problems requiring exponential time(preliminary report). In *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*, STOC '73, page 1–9. Association for Computing Machinery.

Weihao Su, Hong Huang, Rongchen Li, Haiming Chen, and Tingjian Ge. 2024. Towards an effective method of ReDoS detection for non-backtracking engines. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 271–288.

Jiashuo Sun, Yi Luo, Yeyun Gong, Chen Lin, Yelong Shen, Jian Guo, and Nan Duan. 2024. Enhancing chain-of-thoughts prompting with iterative bootstrapping in large language models. In *Findings of the Association for Computational Linguistics: NAACL*, pages 4074–4101.

Sicheol Sung, Hyunjoon Cheon, and Yo-Sub Han. 2022. How to settle the ReDoS problem: Back to the classical automata theory. In *Implementation and Application of Automata - 26th International Conference, CIAA, Proceedings*, volume 13266, pages 34–49.

Xunzhu Tang, Jiechao Gao, Jin Xu, Tiezhu Sun, Yewei Song, Saad Ezzini, Wendkûuni C. Ouédraogo, Jacques Klein, and Tegawendé F. Bissyandé. 2025. SynFix: Dependency-aware program repair via relationgraph analysis. In *Findings of the Association for Computational Linguistics, ACL*, pages 4878–4894.

Lenka Turonová, Lukás Holík, Ivan Homoliak, Ondrej Lengál, Margus Veanes, and Tomás Vojnar. 2022. Counting in regexes considered harmful: Exposing ReDoS vulnerability of nonbacktracking matchers. In *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, pages 4165–4182. USENIX Association.

Brink van der Merwe, Nicolaas Weideman, and Martin Berglund. 2017. Turning evil regexes harmless. In *Proceedings of the South African Institute of Computer Scientists and Information Technologists*, SAICSIT '17, New York, NY, USA. Association for Computing Machinery.

Xinyi Wang, Cen Zhang, Yeting Li, Zhiwu Xu, Shuailin Huang, Yi Liu, Yican Yao, Yang Xiao, Yanyan Zou, Yang Liu, and 1 others. 2023a. Effective ReDoS detection by principled vulnerability modeling and exploit generation. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 2427–2443. IEEE.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023b. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR*.

Sajana Weerawardhena, Paul Kassianik, Blaine Nelson, Baturay Saglam, Anu Vellore, Aman Priyanshu, Supriti Vijay, Massimo Aufiero, Arthur Goldblatt, Fraser Burch, Ed Li, Jianliang He, Dhruv Kedia, Kojin Oshiba, Zhouran Yang, Yaron Singer, and Amin Karbasi. 2025. Llama-3.1-FoundationAI-SecurityLLM-8B-Instruct Technical Report. *Preprint*, arXiv:2508.01059.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. 2022a. Emergent abilities of large language models. *Transactions on Machine Learning Research*, 2022.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022b. Chain-of-Thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 40 others. 2025. Qwen3 technical report.

Yao Yao, Zuchao Li, and Hai Zhao. 2024. GoT: Effective graph-of-thought reasoning in language models. In *Findings of the Association for Computational Linguistics: NAACL*, pages 2901–2921.

Daoguang Zan, Bei Chen, Fengji Zhang, Dianjie Lu, Bingchao Wu, Bei Guan, Yongji Wang, and Jian-Guang Lou. 2023. Large language models meet nl2code: A survey. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL*, pages 7443–7464.

Zhaowei Zhang, Hongyu Zhang, Beijun Shen, and Xiaodong Gu. 2022. Diet code is healthy: simplifying programs for pre-trained models of of code. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE*, pages 1073–1084.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V. Le, and Ed H. Chi. 2023. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations, ICLR*.

A Experimental details

During our experiments, LLM inferences are executed on systems running the Rocky Linux 9.4 operating system, leveraging an NVIDIA RTX A6000 graphic processing unit. RegexScalpel and ReDoSHunter are allocated to a Rocky Linux 8.10 system, which is powered by an AMD Ryzen Threadripper 3960X 24-core processor (48 threads). Regarding the software stack, both RegexScalpel and ReDoSHunter are built using Java openjdk 25, while the core implementation of the LRR framework itself was written in Python 3.12.

B Prompts

The following are prompts employed to configure the LLMs within the LRR framework.

Basic prompt

You are an expert Application Security Analyst specializing in regular expression security.
...
Your entire output should be only the final, corrected regular expression, enclosed in a code block.
For example, if the input is: `(?:a+)+`
your output must be: `a+`
Analyze the following pattern: `{pattern}`

5-shot prompt

`{Basic prompt}`

Here is an example of the repairing process.
`{Five examples of repairing process}`

Analyze the following pattern: `{pattern}`

Localization prompt

`{Basic prompt}`

You must focus on the following subpatterns: `{subpatterns}`. At least one of these subpattern causes catastrophic backtracking.

Analyze the following pattern; you have to answer the repaired version of the entire pattern, not only a repaired subpattern: `{pattern}`

LRR prompt

`{Basic prompt}`

Here is an example of the repairing process.
`{Five examples of repairing process}`

You must focus on the following subpatterns: `{subpatterns}`. At least one of these subpattern causes catastrophic backtracking.

Analyze the following pattern; you have to answer the repaired version of the entire pattern, not only a repaired subpattern: `{pattern}`

C Vulnerability elimination rate

We report the vulnerability elimination rate separately to provide clarity on the model's ability to produce safe regexes, regardless of semantic correctness. Table 4 shows the explicit difference between the two success criteria.

Model	Repair	Vuln. elim.
RegexScalpel	43.60	50.00
Qwen3 (NR)	51.60	61.30
Llama3.1	50.00	65.10
Gemma 3	47.80	55.30
Phi4	**59.00**	**66.60**
Qwen3	51.00	58.60
DS-R1-Qwen	45.50	54.00
GPT-oss-20B (low)	47.50	54.60
GPT-oss-20B (med)	47.40	54.60
GPT5	49.20	55.70

Table 4: Comparison between repair rate and vulnerability elimination rate.

While presenting these raw numbers offers a more complete picture, we maintain that the vulnerability elimination rate alone is an insufficient measure of success. This metric incorrectly classifies repairs as successful even when the pattern is semantically broken (e.g., matching an empty language). Since such repairs render the regex unusable, we exclude them to ensure our primary repair rate metric reflects only practically viable fixes.

D Failure of baseline methods

In this section, we analysis failure cases of rule-based baseline methods.

D.1 RegexScalpel

RegexScalpel (Li et al., 2022) uses negative lookaheads `(?!...)`, which assert that the current position in the string is not followed by a substring that matches the pattern `r`, to remove overlapping patterns. This effectively removes the ReDoS vulnerability by eliminating unambiguous choices during matching. However, if the pattern `r` is chosen too broadly, the resulting overall pattern becomes too narrow. As a result, of the 500 ReDoS-invulnerable regexes generated by RegexScalpel, 39 exhibited a Jaccard similarity of zero when compared to their original patterns.

The following repair example results in an empty language pattern, which cannot match any strings; refer to the rule τ_{33} from Table 8 in Li et al. (2022).

Failure example of RegexScalpel

Original:
```
<(named-content.*?)>
```
Repaired:
```
(?!.*?)<(named-content.*?)>
```

D.2 Remedy

As shown in Table 1, Remedy (Chida and Terauchi, 2022) exhibits a low ratio of well-formed repaired regexes. This occurs because Remedy does not fully support common special symbols frequently found in real-world patterns, such as character classes like `\d` (digits) and `\w` (word characters).

The following failure example illustrates this limitation: the presence of the `\d` symbol, which represents digits, causes Remedy to fail compilation or raise an internal error, resulting in an ill-formed output and contributing to its low overall performance score.

Failure example of Remedy

Original:
```
^POINT\\((-?\\d+\\.?\\d*) (-?\\d+\\.?\\d*)\\$
↪ Remedy raises an error
```

E Behavioral differences between models

The following are selected examples which highlight the difference between non-reasoning (Phi4) and reasoning (Qwen3) models. They utilize two strategies:

(a) match-all symbol ‘`.`’
→ character class ‘`[...]`’

(b) unbounded repetition ‘`*`’ and ‘`+`’
→ bounded repetition ‘`{n,m}`’

This distinct strategic behavior results in the non-reasoning model applying (a), (a, b), and (a, b) sequences, while the reasoning model uses (a), (b), and (b), yielding a more syntactically similar repair.

Repair with different syntactics

Original:
```
^\n*(.*?)\s*$
```
Repaired by Phi4:
```
^\n{0,1000}(.{0,1000})\s{0,1000}$
```
Repaired by Qwen3:
```
^\n*[\s\S]*\s*$
```

Original:
```
image:: (.*\.png)
```
Repaired by Phi4:
```
image:: ([\u0000-...-\u007f]{0,1000}\.png)
```
Repaired by Qwen3:
```
image:: (.{0,500}\.png)
```

Original:
```
/i18n/(.+)\.json$
```
Repaired by Phi4:
```
/i18n/([\u0000-\uFFFF]{1,100})\.json
```
Repaired by Qwen3:
```
/i18n/(.{0,500})\.json$
```

F Full experimental results

Table 1 from Section 6 demonstrates our main results. This section provides the full results. of LRR across all our base LLMs. Tables 5 and 6 present the performance of LRR with non-reasoning and reasoning LLMs, respectively. For each model, we report the performance of our full framework, which combines vulnerability localization with 5-shot CoT prompting. We also include results from three variants: (-) localization, (-) 5-shot, and (-) localization & 5-shot. to demonstrate the individual contributions of each component. Likewise to Section 6, the evaluation metrics cover the rate of well-formed outputs, repair success rate, syntactic similarity (RLI, NLS), and semantic similarity (Jaccard, precision, recall). The results confirm that our observation in Section 6.3 is consistent across LLMs.

Model	Well-formed	Repair	Syntactic similarity		Semantic similarity		
			RLI	NLS	Jaccard	Precision	Recall
Qwen3-8B (NR)	98.60	51.60	79.75	69.93	79.19	92.53	80.83
(-) localization	98.70	50.70	66.29	73.88	80.00	93.16	81.14
(-) 5-shot	99.40	50.40	51.98	75.20	77.08	91.14	79.03
(-) local. & 5-shot	99.20	49.30	52.61	75.16	77.36	91.11	79.32
Llama3.1-8B	96.80	50.00	119.97	58.52	73.06	85.08	74.41
(-) localization	98.00	48.80	103.20	68.79	79.32	91.21	80.73
(-) 5-shot	95.60	48.50	58.26	72.25	75.92	87.93	78.06
(-) local. & 5-shot	95.80	48.10	61.61	73.22	75.93	89.06	77.85
Gemma3-12B	99.40	47.80	63.27	76.68	83.10	93.27	84.87
(-) localization	99.40	46.10	54.69	80.18	84.96	94.17	86.38
(-) 5-shot	99.40	46.10	54.00	77.79	81.49	91.45	83.61
(-) local. & 5-shot	99.60	45.90	53.80	78.91	82.24	92.22	84.22
Phi4-14B	99.30	59.00	100.43	63.37	84.22	95.78	85.93
(-) localization	99.60	57.80	93.94	65.72	83.61	95.13	85.44
(-) 5-shot	97.90	49.70	62.24	72.40	76.91	89.55	79.51
(-) local. & 5-shot	97.40	49.00	62.38	73.05	78.88	90.58	81.31

Table 5: Full experimental results of LRR on non-reasoning large language models.

Model	Well-formed	Repair	Syntactic similarity		Semantic similarity		
			RLI	NLS	Jaccard	Precision	Recall
Qwen3-8B	97.90	51.00	68.79	73.33	83.03	94.47	84.23
(-) localization	98.20	46.60	57.24	77.10	83.61	94.62	84.36
(-) 5-shot	97.90	47.60	53.61	75.39	79.11	91.64	80.84
(-) local. & 5-shot	97.40	47.30	54.51	75.74	79.58	91.41	80.76
DS-R1-Qwen-7B	80.70	45.60	72.44	59.34	63.86	75.03	64.86
(-) localization	74.20	44.80	76.62	55.55	61.92	71.20	62.55
(-) 5-shot	75.20	45.60	72.93	55.54	60.66	70.80	61.51
(-) local. & 5-shot	73.50	45.30	74.01	54.14	60.76	69.73	61.62
GPT-oss-20B (low)	91.00	47.50	63.02	69.41	78.89	88.60	79.64
(-) localization	89.90	46.40	62.79	70.27	79.55	88.22	80.04
(-) 5-shot	98.30	46.70	57.61	76.41	85.21	95.14	86.28
(-) local. & 5-shot	97.70	45.80	56.49	78.71	86.63	95.49	87.14
GPT-oss-20B (med.)	96.80	47.40	61.11	74.45	84.18	94.68	84.82
(-) localization	85.90	44.90	63.64	67.87	76.62	84.64	76.94
(-) 5-shot	97.10	46.60	59.17	75.63	83.18	94.62	83.71
(-) local. & 5-shot	98.10	45.60	55.75	79.39	87.65	96.79	88.08
GPT-5	99.70	49.20	62.91	75.57	89.77	98.96	90.24
(-) localization	99.90	48.30	57.77	79.33	90.54	99.56	90.73
(-) 5-shot	99.50	47.30	60.30	77.30	89.54	98.98	89.80
(-) local. & 5-shot	100.00	46.20	56.17	81.04	91.33	99.48	91.53

Table 6: Full experimental results of LRR on reasoning large language models.