Query4Regex: Verifiable Regex Transformation through Formal Operations from NL and DSL Queries

Joonghyuk Hahn **Yo-Sub Han***
Department of Computer Science, Yonsei University, Seoul, Republic of Korea
{greghahn,emmous}@yonsei.ac.kr

Abstract

While large language models (LLMs) excel at generating structured data, such as code, their ability to precisely manipulate it based on instructions remains relatively under-explored. Regular expressions (regexes), critical in practice, are challenging to manipulate. Crucially, the correctness of transformations can be mathematically verified, making them exceptionally well-suited for measuring the symbolic reasoning of LLMs. We introduce **Query4Regex**, a new benchmark for evaluating verifiable transformations on regexes. Our benchmark tests two query formats: natural language instructions and a program-like domain-specific language (DSL) that specifies the sequence of operations. We evaluate a range of LLMs, verifying semantic correctness through rigorous deterministic finite automata (DFA) equivalence testing. Our empirical studies reveal: 1) the formal DSL significantly outperforms natural language, achieving up to 6.74%p accuracy gains on average. 2) Performance for both formats degrades sharply as compositional complexity increases, highlighting a core challenge in multi-step reasoning. 3) Models often generate plausible but unparsable outputs. Even among parsable outputs, semantic errors remain common, making failures difficult to detect without formal verification. Query4Regex provides a robust framework for analyzing the gap between LLMs' linguistic fluency and their symbolic reasoning, paving the way for more reliable and verifiable manipulation of formal languages. Our code is available at https://github.com/peer0/Query4Regex.

1 Introduction

Large language models (LLMs) have revolutionized the landscape of software development (Wang et al., 2023) through their capabilities of in-context learning (ICL) (Brown et al., 2020; Min et al., 2022; Wei et al., 2022; Madaan et al., 2023). They demonstrate a remarkable ability to generate structured data such as source code from natural language prompts (Mishra et al., 2023; Hou et al., 2024; Sivakumar et al., 2024). Tools such as GitHub Copilot[1] (Chen et al., 2021) have become integral to the modern developer's workflow, showcasing LLMs' proficiency in understanding programmatic intent and producing syntactically correct and contextually relevant code snippets. This success, however, is centered on the task of generation.

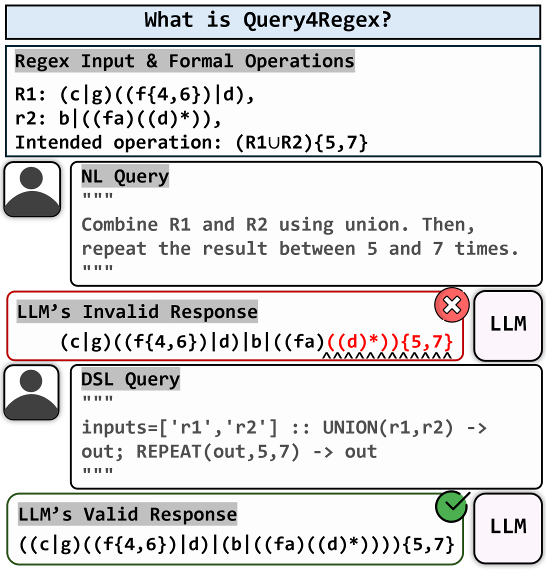

Figure 1: Overview of the Query4Regex task. The benchmark compares LLM performance on natural language and DSL queries for regex manipulation.

While LLMs excel at creating code from scratch (Zan et al., 2023; Zheng et al., 2024), their ability to precisely manipulate existing, often complex, structured data based on high-level instructions remains a relatively under-explored and challenging frontier. Unlike generation, manipulation requires a fundamental understanding of both the

*Corresponding author.

[1] https://copilot.github.com

Findings of the Association for Computational Linguistics: EACL 2026, pages 6297–6305
March 24-29, 2026 ©2026 Association for Computational Linguistics

instruction's semantics and the formal syntax of the targeted data, demanding a high degree of precision and reliability, where a single error can invalidate the entire structure.

We focus on regular expressions (regexes) as an ideal domain to investigate this challenge. While critical in practice for tasks such as text processing and pattern matching (Thompson, 1968), regexes are notoriously challenging for humans to write and manipulate due to their dense syntax and non-intuitive behavior. Crucially, the correctness of regex transformations can be mathematically verified through mechanisms like deterministic finite automata (DFA) equivalence. This verifiability makes them an exceptionally well-suited testbed for rigorously measuring the symbolic reasoning capabilities of LLMs (Liu et al., 2025), moving beyond simple syntactic correctness to true semantic equivalence.

To this end, we introduce **Query4Regex**, a new benchmark and task designed to evaluate the verifiable transformation of regexes systematically. Our benchmark requires models to modify a given regex based on a high-level query. We test two distinct query formats: human-centric natural language instructions and domain-specific language (DSL) instructions that formally specify the sequence of operations.

We scope Query4Regex to *classical* regular-expression operators—union, intersection, concatenation, complement, Kleene star, and bounded repetition—as defined in automata theory (Hopcroft et al., 2006). This choice keeps the task within regular languages, enabling exact verification via deterministic finite automata equivalence. Although practical engines support richer constructs (e.g., lookarounds and backreferences), large-scale ecosystem measurements report that such features occur in only a small fraction of real-world regexes (e.g., <5%) (Davis et al., 2018; Wang and Stolee, 2018; Davis et al., 2019).

2 Related Works

Our work is positioned as structured data manipulation with LLMs. The remarkable success of LLMs in software engineering is well-documented (Fan et al., 2023; Pan et al., 2024), demonstrating strong capabilities in code generation or completion from natural language (Bai et al., 2025; Black et al., 2025). Subsequent research has extended these capabilities to tasks such as automated program repair and bug detection (Joshi et al., 2023; Chen et al., 2024; Wang et al., 2024; Wu et al., 2024). However, these studies predominantly focus on generating functionally correct code in general-purpose languages, where correctness is often evaluated through unit tests (Yang et al., 2024a). In contrast, our work centers on the precise, step-by-step transformations of a specialized formal language structure. We introduce a stricter criterion of verifiable semantic equivalence.

The task of generating regexes from natural language has also been a long-standing area of research (Park et al., 2019; Ye et al., 2020a,b). The dominant paradigm in this area is generation from scratch, where a model synthesizes a regex from a semantic description of a target pattern (Ye et al., 2021; Mao et al., 2023). Query4Regex introduces a fundamentally different task: the transformation of existing regexes based on procedural and operational instructions. This requires the model not only to parse semantics but also to understand and execute a sequence of formal operations, directly testing its compositional reasoning abilities in a way that generation from scratch does not.

Our work contributes to the growing body of research evaluating the symbolic and algorithmic reasoning capabilities of LLMs (Li et al., 2024; Morishita et al., 2024; Petruzzellis et al., 2024; Yang et al., 2024b), which often lie beyond their core linguistic competencies. While existing benchmarks have made significant progress in evaluating mathematical and logical reasoning (Stolfo et al., 2023; Wu et al., 2023; Rai and Yao, 2024; Zhang et al., 2024), they do not typically focus on the direct manipulation of structures with strict, formal syntax. Query4Regex provides a novel testbed for this specific form of reasoning, probing the ability of LLMs to bridge the gap between understanding a linguistic command and executing a symbolic manipulation on a regex.

3 Query4Regex

3.1 Problem Definition

The task of verifiable regex transformation is to generate a target regex r_t that is semantically equivalent to the result of applying a series of formal operations, specified by a query q, to a set of source regexes $R = \{r_1, \ldots, r_n\}$. The query q is in one of two formats: a natural language instruction q_{NL} or a formal DSL expression q_{DSL}. The generated regex r_p must be semantically equivalent to the ground-truth target regex r_t. We verify this condition through DFA equivalence, $r_p \equiv_{DFA} r_t$.

3.2 Dataset Construction

Our benchmark is constructed through a systematic, multi-step pipeline designed to ensure correctness and control for complexity. The process is as follows:

1. Seed generation: We first generate a set of simple, atomic regexes through random sampling to use as the initial inputs $R = \{r_1, r_2, \ldots\}$.

2. Formal operation synthesis: We randomly sample a sequence of one to three formal operations. The operations are union, intersection, concatenation, complement, Kleene star, and numeric repetitions. We apply this sequence of operations to the seed regexes to produce a correct target regex r_t.

3. Query generation: The sequence of operations from the previous step is translated into its corresponding natural language query q_{NL} or DSL query q_{DSL}. We checked over q_{NL} and q_{DSL} to ensure they represent the exact same underlying transformations.

3.3 Query Scenarios

Our benchmark, Query4Regex, evaluates the capabilities of LLMs on two distinct but semantically equivalent query scenarios: natural language instructions (q_{NL}) and formal DSL instructions (q_{DSL}). Table 6 in Appendix A illustrates the key differences between these two scenarios.

Natural language queries are designed to reflect how a human might specify a transformation, using flexible and potentially ambiguous representations such as “the result” or “it”. This format tests the model’s ability to infer intent from informal language. In contrast, the DSL provides an unambiguous, machine-parsable representation of the formal operation transformation. Its syntax, defined by `inputs :: op1` → `var1; op2` → `var2;` ..., explicitly defines the inputs and the sequence of operations with the flow of intermediate results. This format directly tests the model’s ability to follow a formal, procedural specification.

We test these queries under two prompting settings. In the zero-shot setting, the model receives only the instruction and the source regexes. This assesses the model’s intrinsic, pre-trained knowledge of regex operations. In the five-shot setting, the prompt includes five transformation examples before the actual test instances. This evaluates the model’s in-context learning ability to recognize and replicate the required reasoning patterns. We randomly sample these examples from an auxiliary pool generated by the same pipeline, disjoint from the 1,000 evaluation instances.

4 Evaluation Settings

We detail the experimental setup for evaluating the LLMs on the Query4Regex benchmark. We describe the models, the data configuration, and the metrics for evaluation.

4.1 Base LLMs

We select a diverse suite of recent and powerful LLMs to comprehensively assess the capabilities for regex manipulation. We cover a wide range of architectures and parameter scales. The models in our empirical studies include: `gemma-3` (12B, 27B) (Kamath et al., 2025), `gpt-oss` (20B, 120B) (Agarwal et al., 2025), `Phi-4` (14B), `Phi-4-reasoning` (14B) (Abdin et al., 2024), `Llama-3.3` (70B) (Dubey et al., 2024).

4.2 Data used for Evaluation

Query4Regex consists of 1,000 unique instances of regex transformation tasks. As shown in Table 1, this dataset is composed of queries with varying levels of complexity, from single-step formal operations to more complex three-step sequences.

# of Operations	1	2	3
# of Instances	671	182	147
Percentage	67.1%	18.2%	14.7%

Table 1: Distribution of operational complexity in the Query4Regex dataset.

4.3 Evaluation Metrics

Multiple representations exist for the same regex. For instance, `a|b` is equivalent to `b|a`. Simple string comparison is not sufficient to measure the correctness and thus, we use the following metrics for a rigorous evaluation.

Semantic equivalence. This determines whether the generated regex r_p and the target regex r_t accept the exact same language. We verify this through

Model	q_{NL}						q_{DSL}					
	Zero-shot			Five-shot			Zero-shot			Five-shot		
	Syn.	**Sem.**	**Sem.**†	**Syn.**	**Sem.**	**Sem.**†	**Syn.**	**Sem.**	**Sem.**†	**Syn.**	**Sem.**	**Sem.**†
Phi-4 (14B)	44.1	16.7	37.87	44.6	21.2	47.53	44.6	13.5	30.27	44.3	18.0	40.63
gemma-3 (27B)	44.8	21.7	48.44	46.0	23.1	50.22	44.2	23.3	52.71	44.4	20.2	45.50
Llama-3.3 (70B)	43.1	16.2	37.59	43.5	19.4	44.60	42.2	16.3	38.63	43.4	21.1	48.62
Phi-4-reasoning (14B)	35.28	16.28	46.15	33.92	17.38	51.24	46.6	27.6	59.23	46.1	**29.2**	**63.34**
gpt-oss (20B)	45.2	16.38	36.24	47.0	19.76	42.04	**47.9**	23.4	48.85	46.9	28.2	60.13
gpt-oss (120B)	47.1	15.4	32.22	45.3	16.4	36.20	47.8	19.2	40.76	45.3	28.3	62.47

Table 2: Overall performance on Query4Regex across two query scenarios (q_{NL} and q_{DSL} and two prompting settings (zero-,five-shot). **Syn.** denotes syntactic correctness, **Sem.** denotes semantic equivalence. **Sem.**† is the semantic equivalence conditioned on syntactically correct outputs. Phi-4, gemma-3, and Llama-3.3 are non-reasoning LLMs. Phi-4-reasoning and gpt-oss are reasoning LLMs.

Model	q_{NL}—Zero-shot						q_{NL}—Five-shot					
	Ops			Filtered Ops			Ops			Filtered Ops		
	1	2	3	1	2	3	1	2	3	1	2	3
Phi-4 (14B)	19.37	12.64	9.52	41.67	30.67	25.93	23.70	19.78	11.56	49.84	46.15	34.69
gemma-3 (27B)	26.23	16.48	7.48	54.49	42.25	20.37	29.66	12.09	6.80	59.76	29.73	18.87
Llama-3.3 (70B)	19.37	12.09	6.80	41.94	32.35	18.87	22.95	17.03	6.12	48.73	43.66	18.75
Phi-4-reasoning (14B)	18.03	17.52	6.73	51.10	49.67	19.08	21.64	9.67	7.48	63.79	28.51	22.05
gpt-oss (20B)	23.50	18.00	0.00	51.99	39.82	0.00	20.15	15.55	0.68	42.87	33.09	1.45
gpt-oss (120B)	19.52	9.89	3.40	39.10	21.69	8.33	18.03	13.74	12.24	37.35	32.05	35.29

Table 3: Per-operation (1 to 3) semantic equivalence scores for the q_{NL} scenario. **Filtered Ops** refers to semantic equivalence on syntactically correct outputs only.

DFA equivalence testing.

$$\text{Semantic Equivalence} = \frac{|\{r_p \mid r_p \equiv_{DFA} r_t\}|}{\text{Total \# of instances}}.$$

Syntactic Correctness. We also report the validity of a model output by evaluating whether the given output is a valid parsable regex. This metric helps distinguish between models that produce structured but flawed outputs versus those that fail by generating completely malformed strings.

$$\text{Syntactic Correctness} = \frac{|\{r_p \mid r_p \text{ is valid}\}|}{\text{Total \# of instances}}.$$

5 Empirical Studies

We present the results of our empirical evaluation on three non-reasoning LLMs: Phi-4, gemma-3, Llama-3.3, and three reasoning LLMs: Phi-4-reasoning, gpt-oss (20B, 120B). The results indicate that q_{DSL} is more effective than q_{NL} in regex manipulation.

5.1 Natural Language vs. Formal DSL

Table 2 demonstrates that q_{DSL} enables substantially higher semantic correctness than q_{NL}, particularly for larger models and those specialized for reasoning. For instance, in the five-shot setting, gpt-oss (120B)'s Sem.† score surges from 36.20% on q_{NL} to 62.47% on q_{DSL}, a 26.3%p gain. This suggests that while models can often generate a structurally valid regex from either query format—as shown by the relatively similar Syn. scores—the ambiguity of natural language frequently leads them to produce semantically incorrect output.

A particularly notable finding is that Phi-4-reasoning (14B) achieves the highest overall semantic equivalence, outperforming much larger gpt-oss (120B). This indicates that the model architecture and pre-trained data for reasoning can be more critical than raw parameter scale alone. We further investigate the performance of Phi-4-reasoning (14B) and gpt-oss (120B) in Section 5.2.

5.2 Impact of Compositional Complexity

We further analyze semantic equivalence by the number of sequential operations (1–3) across both query formats and prompting settings. For Tables 3 and 4, the **Ops** columns show the overall semantic equivalence score, while the **Filtered Ops** columns show the semantic equivalence conditioned only on syntactically correct outputs, providing insight

Model	q_{DSL}—Zero-shot						q_{DSL}—Five-shot					
	Ops			Filtered Ops			Ops			Filtered Ops		
	1	2	3	1	2	3	1	2	3	1	2	3
Phi-4 (14B)	14.90	13.74	6.80	31.45	32.89	19.23	22.06	13.74	4.76	46.25	34.25	14.00
gemma-3 (27B)	27.42	19.78	8.84	59.16	46.75	24.07	23.99	17.03	6.80	50.31	40.79	20.83
Llama-3.3 (70B)	19.08	14.84	5.44	41.03	40.30	18.60	23.99	19.23	10.20	50.79	52.24	30.00
Phi-4-reasoning (14B)	32.64	21.43	12.24	65.96	48.75	33.33	35.32	19.78	12.93	70.75	46.75	38.78
gpt-oss (20B)	28.61	18.13	6.12	57.14	38.82	15.52	33.23	22.53	12.24	67.78	50.00	31.03
gpt-oss (120B)	24.14	12.64	4.76	48.94	27.71	12.28	32.49	24.73	13.61	66.87	59.21	39.22

Table 4: Per-operation (1 to 3) semantic equivalence scores for the q_{DSL} scenario. **Filtered Ops** refers to semantic equivalence on syntactically correct outputs only.

into the model's accuracy when it successfully generates a valid regex.

Table 3 details the per-operation performance for tasks guided by natural language queries. The results clearly show a steep degradation in performance for all models as the number of operations increases from 1 to 3. While the five-shot setting generally provides a performance uplift for simpler, single-operation tasks, it does not substantially mitigate the performance collapse on more complex, multi-step transformations, highlighting the inherent difficulty models face in parsing and executing ambiguous, compositional instructions.

Table 4 shows the per-operation performance when models are guided by the formal DSL queries. Similar to the natural language scenario, a performance degradation is observed as complexity increases. However, the baseline performance, especially for single-operation tasks and for reasoning-oriented models such as Phi-4-reasoning, is notably higher than their q_{NL} counterparts. This reinforces the conclusion from our main analysis that the unambiguous structure of the DSL provides a more effective signal for models to ground their reasoning, though it does not fully solve the core challenge of multi-step, compositional logic.

Error type	Phi-4-reasoning (14B)	gpt-oss (120B)
Correct	292	283
Unparsable (Syntax)	539	547
Semantic error	169	170

Table 5: Error taxonomy on the q_{DSL}, five-shot scenario. Unparsable outputs fail regex parsing; semantic errors are syntactically valid but fail DFA equivalence.

5.3 Error Analysis

We analyze failures in the best-performing setting (q_{DSL}, five-shot) to contextualize the low semantic-equivalence scores. Table 5 shows that most errors are syntactic. Models frequently output unparsable regexes, which are over half of the total instances, despite producing plausible structures. Among parsable outputs, a substantial fraction remains semantically incorrect, underscoring the need for formal verification beyond syntax.

We observe two dominant failure modes in semantic errors. First is about scope leakage in multi-step instructions, such as "concatenate r1 and r2, then apply Kleene star to the result". This suggests difficulty in maintaining the scope and the order of operators, especially when given multiple operations. Second is partial correctness. Models sometimes ignore the final operation in sequences such as "`out = UNION(r1, r2); out = INTERSECTION(out, r3)`". Such patterns indicate that maintaining intermediate symbolic state remains challenging even under unambiguous procedural specifications.

6 Conclusion

We introduce Query4Regex, a new benchmark to evaluate the ability of LLMs to perform verifiable transformations on regexes. Our empirical studies yield two key findings: First, the unambiguous, formal DSL is significantly more effective than natural language for regex manipulation, particularly for reasoning LLMs. Second, all models exhibit a sharp decline in performance as the number of formal operations increases, highlighting a fundamental weakness in compositional reasoning. Our work provides a robust framework for probing the gap between linguistic fluency and symbolic reasoning of LLMs in the regex manipulation task.

Limitations

Our benchmark is scoped to the core set of formal regular expression operations defined in Hopcroft et al. (2006), excluding complex practical features such as lookarounds and backreferences whose formal verification is non-trivial. It remains a meaningful open question on the performance of LLMs on these practical regex features. The evaluation is conducted on 1,000 instances with up to three operations, a scale that represents a trade-off between analytical depth and the significant computational cost of inference. While larger and more complex datasets could yield further insights, we find our current benchmark is sufficient for robustly evaluating the core compositional reasoning of LLMs for regex manipulation. We plan to release a larger Query4Regex benchmark with more complex features.

Acknowledgment

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*, CoRR 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.

Xingyuan Bai, Shaobin Huang, Chi Wei, and Rui Wang. 2025. Collaboration between intelligent agents and large language models: A novel approach for enhancing code generation capability. *Expert Systems with Applications*, 269:126357.

Gavin Black, Bhaskar Prasad Rimal, and Varghese Mathew Vaidyan. 2025. Balancing security and correctness in code generation: An empirical study on commercial large language models. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 9(1):419–430.

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)*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *ArXiV Preprint*, CoRR 2107.03374.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2024. Teaching large language models to self-debug. In *The Twelfth International Conference on Learning Representations, ICLR*.

James C. Davis, Christy A. Coghlan, Francisco Servant, and Dongyoon Lee. 2018. The impact of regular expression denial of service (ReDoS) in practice: an empirical study at the ecosystem scale. In *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE*, pages 246–256.

James C. Davis, Daniel Moyer, Ayaan M. Kazerouni, and Dongyoon Lee. 2019. Testing regex generalizability and its implications: A large-scale many-language measurement study. In *34th IEEE/ACM International Conference on Automated Software Engineering, ASE*, pages 427–439.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, and 82 others. 2024. The llama 3 herd of models. *ArXiv Preprint*, CoRR 2407.21783.

Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M. Zhang. 2023. Large language models for software engineering: Survey and open problems. In *IEEE/ACM International Conference on Software Engineering: Future of Software Engineering, ICSE-FoSE*, pages 31–53.

John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. 2006. *Introduction to Automata Theory, Languages, and Computation (3rd ed.)*. Pearson Education, Inc., Boston, MA, USA.

Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology*, 33(8):220:1–220:79.

Harshit Joshi, José Pablo Cambronero Sánchez, Sumit Gulwani, Vu Le, Gust Verbruggen, and Ivan Radicek. 2023. Repair is nearly generation: Multilingual program repair with llms. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI*, pages 5131–5140.

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. *ArXiV Preprint*, CoRR 2503.19786.

Zhaoyi Li, Gangwei Jiang, Hong Xie, Linqi Song, Defu Lian, and Ying Wei. 2024. Understanding and patching compositional reasoning in llms. In *Findings of the Association for Computational Linguistics, ACL*, pages 9668–9688.

Chengwu Liu, Ye Yuan, Yichun Yin, Yan Xu, Xin Xu, Zaoyu Chen, Yasheng Wang, Lifeng Shang, Qun Liu, and Ming Zhang. 2025. Safe: Enhancing mathematical reasoning in large language models via retrospective step-aware formal verification. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL*, pages 12171–12186.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems (NeurIPS)*.

Chenhui Mao, Xiexiong Lin, Xin Jin, and Xin Zhang. 2023. Enhancing language model with unit test techniques for efficient regular expression generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: EMNLP*, pages 12–19.

Sewon Min, Mike Lewis, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2022. 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.

Mayank Mishra, Prince Kumar, Riyaz A. Bhat, Rudra Murthy V, Danish Contractor, and Srikanth Tamilselvam. 2023. Prompting with pseudo-code instructions. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 15178–15197.

Terufumi Morishita, Gaku Morio, Atsuki Yamaguchi, and Yasuhiro Sogawa. 2024. Enhancing reasoning capabilities of llms via principled synthetic logic corpus. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems (NeurIPS)*.

Rangeet Pan, Ali Reza Ibrahimzada, Rahul Krishna, Divya Sankar, Lambert Pouguem Wassi, Michele Merler, Boris Sobolev, Raju Pavuluri, Saurabh Sinha, and Reyhaneh Jabbarvand. 2024. Lost in translation: A study of bugs introduced by large language models while translating code. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE*, pages 82:1–82:13.

Jun-U. Park, Sang-Ki Ko, Marco Cognetta, and Yo-Sub Han. 2019. Softregex: Generating regex from natural language descriptions using softened regex equivalence. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP*, pages 6424–6430.

Flavio Petruzzellis, Alberto Testolin, and Alessandro Sperduti. 2024. Benchmarking GPT-4 on algorithmic problems: A systematic evaluation of prompting strategies. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation, LREC/COLING*, pages 2161–2177.

Daking Rai and Ziyu Yao. 2024. An investigation of neuron activation as a unified lens to explain chain-of-thought eliciting arithmetic reasoning of llms. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL*, pages 7174–7193.

Mithila Sivakumar, Alvine Boaye Belle, Jinjun Shan, and Kimya Khakzad Shahandashti. 2024. Prompting GPT-4 to support automatic safety case generation. *Expert Systems with Applications*, 255:124653.

Alessandro Stolfo, Yonatan Belinkov, and Mrinmaya Sachan. 2023. A mechanistic interpretation of arithmetic reasoning in language models using causal mediation analysis. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 7035–7052.

Ken Thompson. 1968. Regular expression search algorithm. *Communications of the ACM*, 11(6):419–422.

Chengpeng Wang, Wuqi Zhang, Zian Su, Xiangzhe Xu, and Xiangyu Zhang. 2024. Sanitizing large language models in bug detection with data-flow. In *Findings of the Association for Computational Linguistics: EMNLP*, pages 3790–3805.

Peipei Wang and Kathryn T. Stolee. 2018. How well are regular expressions tested in the wild? In *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE*, pages 668–678.

Yue Wang, Hung Le, Akhilesh Gotmare, Nghi D. Q. Bui, Junnan Li, and Steven C. H. Hoi. 2023. CodeT5+: Open code large language models for code understanding and generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 1069–1088.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. 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, NeurIPS*.

Haoze Wu, Clark W. Barrett, and Nina Narodytska. 2024. Lemur: Integrating large language models in automated program verification. In *The Twelfth International Conference on Learning Representations, ICLR*.

Zhengxuan Wu, Atticus Geiger, Thomas Icard, Christopher Potts, and Noah D. Goodman. 2023. Interpretability at scale: Identifying causal mechanisms in alpaca. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems, NeurIPS*.

Lin Yang, Chen Yang, Shutao Gao, Weijing Wang, Bo Wang, Qihao Zhu, Xiao Chu, Jianyi Zhou, Guangtai Liang, Qianxiang Wang, and Junjie Chen. 2024a. On the evaluation of large language models in unit test generation. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering, ASE*, pages 1607–1619.

Sohee Yang, Elena Gribovskaya, Nora Kassner, Mor Geva, and Sebastian Riedel. 2024b. Do large language models latently perform multi-hop reasoning? In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL*, pages 10210–10229.

Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. 2020a. Benchmarking multimodal regex synthesis with complex structures. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL*, pages 6081–6094.

Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. 2021. Optimal neural program synthesis from multimodal specifications. In *Findings of the Association for Computational Linguistics: EMNLP*, pages 1691–1704.

Xi Ye, Qiaochu Chen, Xinyu Wang, Isil Dillig, and Greg Durrett. 2020b. Sketch-driven regular expression generation from natural language and examples. *Transactions of the Association for Computational Linguistics*, 8:679–694.

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.

Wei Zhang, Chaoqun Wan, Yonggang Zhang, Yiu-ming Cheung, Xinmei Tian, Xu Shen, and Jieping Ye. 2024. Interpreting and improving large language models in arithmetic calculation. In *Forty-first International Conference on Machine Learning, ICML*.

Tianyu Zheng, Ge Zhang, Tianhao Shen, Xueling Liu, Bill Yuchen Lin, Jie Fu, Wenhu Chen, and Xiang Yue. 2024. Opencodeinterpreter: Integrating code generation with execution and refinement. In *Findings of the Association for Computational Linguistics, ACL*, pages 12834–12859.

Operation	Natural Language Query (qNL)	Formal DSL Query (qDSL)
CONCAT(r1, r2) then STAR	Take r1 and concatenate it with r2, then apply the Kleene star to the whole result.	inputs=['r1','r2'] :: CONCAT(r1,r2) -> o1; STAR(o1) -> out
UNION(r1, STAR(r2))	Give me a regex that matches either r1, or zero or more repetitions of r2.	inputs=['r1','r2'] :: STAR(r2) -> o1; UNION(r1,o1) -> out

Table 6: Comparison of natural language and DSL query scenarios for an example transformation.

A Query Scenario Example

This section provides concrete examples of the two query formats used in our benchmark: natural language (q_{NL}) and the formal Domain-Specific Language (q_{DSL}). As illustrated in Table 6, both formats can represent the same underlying transformation, but they differ significantly in their structure and level of ambiguity.

The q_{NL} format is designed to mimic human-like, conversational instructions. It often uses flexible phrasing (e.g., "Take r1 and. . . ") and potentially ambiguous anaphoric references (e.g., "the whole result"). This format tests an LLM's ability to infer procedural intent from informal language. In contrast, the q_{DSL} format is a fully unambiguous, machine-parsable specification. Its programmatic syntax explicitly defines the inputs, the sequence of formal operations, and the flow of intermediate results via variables, eliminating any potential for misinterpretation.

B Experimental Details

We outline the specific software, hardware, and libraries used to ensure the reproducibility of our experiments. All our experiments are performed on the Rocky Linux 9.6 operating system, using an NVIDIA A6000 GPU for small-to-medium models and an NVIDIA RTX PRO 6000 Blackwell GPU for the largest models. Our implementation is based on Python 3.11. For formal language operations and the core DFA equivalence testing, we utilize the Pyformlang (v1.0.10) library. The LLMs are loaded and managed using the Hugging Face ecosystem, specifically the Transformers (v4.56.2) and Accelerate (v1.10.1), with PyTorch (v2.8.0) serving as the backend deep learning framework. We employ NF4 quantization through BitsAndBytes (v0.47.0) library to efficiently manage the memory requirements of large-scale models. For all generations, we use a deterministic greedy decoding strategy to ensure that our results are fully reproducible.

C Use of AI Assistants in Writing

We acknowledge the use of Gemini 2.5 Pro as an assistant in preparing this manuscript. The model's role is strictly limited to supporting the authors in refining the original sentences. The AI assistant did not contribute to developing an original research idea, experimental design, or the generation of scientific insights and analyses for Query4Regex. Human authors produce all scientific contributions, and the final manuscript is verified by the human authors, who take full responsibility for the content.