Converting REGEX to LIKE Patterns Using LLM

Abhishek Prajapati - SR Number: 24295
Anish Tiwari - SR Number: 24423

Introduction

- **Objective:** Improve the performance of SQL queries by replacing REGEX with SQL LIKE patterns wherever possible. This requires identifying patterns that can be fully or partially replaced and implementing fallbacks to REGEXP when full replacements are not possible.
- **Motivation:**
  - SQL LIKE is a simpler pattern-matching mechanism compared to REGEXP. It is optimized for performance in most database systems, making it significantly faster for queries involving pattern matching.
  - REGEX, while powerful and flexible, introduces additional computational overhead due to its advanced capabilities such as lookaheads, backreferences, and nested quantifiers. These features make REGEXP queries resource-intensive and slower, especially for large datasets.
  - By replacing REGEX patterns with equivalent LIKE patterns, wherever possible, we can improve query performance without compromising functionality. • **Challenges:**
  - Not all REGEX patterns have direct equivalents in LIKE. For instance, constructs like lookaheads, backreferences, and certain quantifiers cannot be represented with LIKE.
  - Ensuring semantic equivalence during conversion requires careful analysis of REGEX patterns.
  - For complex patterns, hybrid solutions that combine LIKE and REGEXP are necessary.
- **Approach:**
  - Created a manual dataset of approximately 100 REGEX along with their LIKE patterns covering a wide range of constructs and complexities.
  - Created a set of conversion rules to translate REGEX patterns to LIKE syntax. Examples include mapping '.*' to '%' (zero or more characters) and '.' to ' ' (any single character).

- Where full conversion is not possible, evaluated hybrid solutions that use both LIKE and REGEXP.
- Leveraged large language models (LLMs), such as LLaMA, for advanced pattern recognition and conversion using various prompting techniques.
- Explored zero-shot, one-shot, few-shot, short circuiting, hybrid approach with rule based and LLAMA, semantic mapping of regex to LIKE, prompting techniques for converting REGEX patterns to LIKE.

Assumptions

• We have used an SQL Server for our implementation and testing. • The SQL Server supports the following wildcards, which were leveraged for converting REGEX patterns into LIKE patterns wherever applicable:

Symbol	Description
%	Represents zero or more characters.
_	Represents a single character.
[]	Represents any single character within the brackets.
-	Represents any single character within the specified range.

Table 1: Wildcards supported by SQL Server

Technical Accomplishments

1. Rule-Based Conversion System

Developed a set of rules to convert REGEX to LIKE patterns:

REGEX Pattern	LIKE Equivalent	Description
.	_	Any single character
.*	%	Zero or more of any character
.+	%	One or more of any character
[abc]	%[abc]%	Any single character in the set
[a-z]	%[a-z]%	Character range
[^a-z]	NOT LIKE %[a-z]%	Negated character range
^pattern	LIKE 'pattern%'	Start of string
pattern$	LIKE '%pattern'	End of string
^pattern$	LIKE 'pattern'	Exact match
abc|def	LIKE '%abc%' OR LIKE '%def%'	Alternation
a?b	'b' OR 'ab'	Optional character

Table 2: Rule-based REGEX to LIKE conversions.

Handling Repetitions:

- Exact Repetitions ({n}):
  - If $n \leq 10$, repeat the character/expression n times.
- Range Repetitions ({n,m}):
  - If $(m - n + 1) \leq 10$, expand from n to m times using OR conditions.
  - If larger, expand from n to n+9 using OR conditions, and use REGEXP for the rest.

Regex not convertible to LIKE:

- (expression)* - Zero or more repetitions.
- (expression)+ - One or more repetitions.
- Lookaheads and Lookbehinds - These constructs have no direct equivalent in LIKE.
- Backreferences - Not supported in LIKE.
- Nested quantifiers and complex groupings.
- **Solution:** For patterns that cannot be fully converted, we will find shortest possible matching strings of REGEX. Then by the use of SHORT CIRCUITING, result is then used in combination with LIKE for improved performance.

2. Prompting Techniques for Conversion

- Explored various prompting techniques. They are listed below:

1. **Zero-Shot Prompting:** Zero-shot prompting refers to the technique where a model is asked to perform a task without having been explicitly trained on examples for that specific task. In this context, the model is provided with a prompt to convert a REGEX pattern to a LIKE pattern, without prior examples or fine-tuning on similar tasks. The model must rely on its general language understanding and reasoning abilities to perform the conversion. The following results were obtained on doing Zero-Shot Prompting on sample of dataset. **Results:**

INPUT REGEX	LLM's OUTPUT	REMARK
abc	*LIKE'%abc%'*	*Correct*
a∗	*LIKE'%%'*	*Wrong*
a.	*LIKE'a%'*	*Wrong*
.∗	*LIKE'% %'*	*Wrong*
ns	*LIKE'%%'*	*Wrong*
[a − zA − Z0 − 9]	*LIKE'[a − zA − Z0 − 9]'*	*Wrong*

nd	*LIKE'%[0 − 9]%'*	*Correct*
nD	*LIKE'%[^(0 − 9)]%'*	*Correct*
y{1,3}	*LIKE'[y]{1,3}'*	*Wrong*
a+	*LIKE'a%c'*	*Wrong*

2. **One-Shot Prompting:** One-shot prompting involves providing the model with a single example to guide its understanding. For instance, showing the model how to convert the REGEX pattern \d to LIKE '%[0-9]%' helps the model learn

Eg Regex	Corresponding Eg LIKE	Input Regex	LLM's Output
a|*b*	*LIKE'%a%'orLIKE'%b%'*	*a*|*b*|*c*	*LIKE'%a%'ORLIKE'%b%'O*
abc)?	*LIKE'%abc%'orLIKE'%'*	(*ab*)?	*LIKE'%ab%'ORLIK*
a	*LIKE'a%'*	*abc*	*LIKE'abc%'*

how to apply this conversion to similar patterns. This approach is useful when limited examples are available, allowing the model to generalize and perform well with minimal data. The following results were obtained on doing One-Shot Prompting. **Results:**

3. **Few-Shot Prompting:** Few-shot prompting provides the model with multiple examples to demonstrate the task. For example, by showing several REGEX patterns and their corresponding LIKE conversions, the model learns the underlying patterns and can apply this knowledge to new, unseen patterns. This approach enhances the model's ability to generalize and handle more complex conversions with improved accuracy.

4. **Hybrid Approach:** The hybrid approach combines both rule-based methods and language model prompting, like LLaMA. This approach takes advantage of the efficiency and accuracy of predefined rules for simpler conversions, while also tapping into the power of the language model to handle more complex or tricky patterns. By blending these two methods, we can achieve better and more reliable REGEX to LIKE conversions.

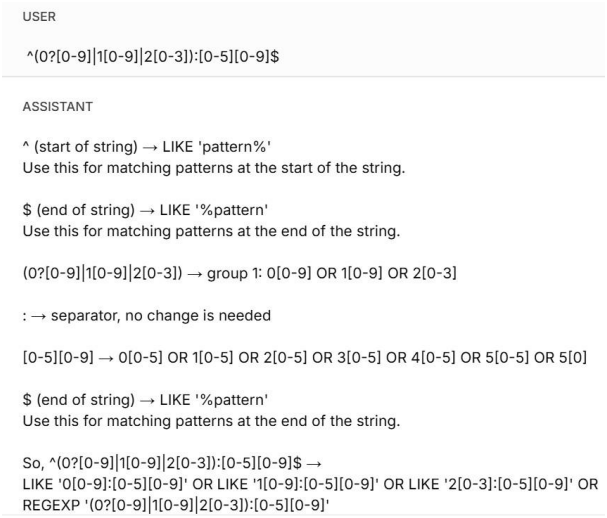

Figure 1: Hybrid Approach Prompt

5. **Short Circuiting:** Short circuiting optimizes the conversion process by focusing on finding the simplest possible `LIKE` patterns first. Instead of evaluating every part of the REGEX, it quickly identifies the shortest matches and applies the conversion rules accordingly. This method helps speed up the process, especially when dealing with large datasets or complex patterns, ensuring that we only apply more advanced techniques when necessary.

6. **Semantic Prompting:**

   Transformer models do not treat tokens (e.g., \d or %) as isolated symbols but instead attempt to understand the context behind them. This makes semantics crucial for regex interpretation, as semantics provide the context and intent that raw regex patterns often lack.

   - **Semantics in Regex Interpretation:** Semantics offer a descriptive and intent-based understanding of regex patterns. For example:
     - **Regex:** \d{3}
     - **Semantic Description:** "A sequence of exactly 3 numeric digits."
   - **Semantic Mapping Tables:** Semantic mapping tables can be employed to bridge the gap between regex patterns and their contextual meanings. These tables contain common regex patterns and their corresponding semantic descriptions. For instance:
     - **Regex:** \d+
     - **Semantic:** "Matches one or more numeric digits."
   - **Context-Aware Mapping:** For complex regex patterns, rules can be built to identify the context in which the regex is used. For example:
     - **Regex:** \d{4}

    - **Possible Contexts:** Represents a year, a date, or a zip code.
  - **Context Identification:** Context can also be inferred from the surrounding text or metadata associated with the regex. This contextual information improves the model's ability to interpret patterns accurately.
  - **Iterative Feedback and Prompting Techniques:** Combining contextaware mapping with iterative feedback-based prompting techniques can significantly enhance the performance of transformer models. Iterative feedback refines results through repeated interactions, ensuring that the model adapts to specific requirements.

3. Dataset Creation and Testing

For dataset creation, we manually compiled a diverse set of around 100 REGEX patterns, including various types commonly used in real-world scenarios. These patterns covered a broad spectrum of use cases, such as:

- Email validation patterns
- URL matching patterns
- Time and date format patterns
- Social Security Number (SSN) validation patterns
- Credit card number patterns
- Password validation patterns
- Quantifier-based patterns (like *, +, ?, {n,m})

This variety of patterns helped test the conversion rules under different complexities and scenarios. Each pattern was designed to explore specific conversion challenges, ensuring we could assess the effectiveness of our approach on a wide range of REGEX types.

We then analyzed the results for each technique applied to the dataset. The findings revealed that certain techniques performed better with specific types of REGEX patterns. For example, patterns like those used for email or URL matching, were converted more accurately using short-circuiting approach, while more complex patterns, especially those with quantifiers or lookbehinds, required fallback to `REGEXP` for accurate matching.

4. Results of Prompting Techniques

Through extensive experimentation with various prompting methodologies, the following observations were made:

- **Prompting Techniques:** One-shot and few-shot prompting techniques showed a significant improvement over zero-shot prompting. However, their performance

was highly sensitive to minor tweaks in the provided examples. Small changes in the structure or content of examples often led to inconsistent results and a drop in accuracy, highlighting the fragility of these approaches.

- **Hybrid Approach:** Employing a hybrid approach, which involves feeding more rules and combining them into prompts, led to a decrease in accuracy. This decline is attributed to the inherent complexity of multiple rules being combined in various ways, creating ambiguities that confused the model. The LLAMA model struggled to generalize effectively under such conditions, emphasizing the need for careful curation of inputs.

- **Semantic Techniques:** Semantic techniques outperformed other methods and provided the best results. Explicitly providing semantic meanings and accurately capturing the intent behind the regex proved critical when working with transformerbased models. By focusing on the purpose and context of regex patterns, these techniques allowed the model to interpret and process patterns more effectively.

  The accuracy of various prompting techniques is summarized below:

  - **Zero Shot Prompting:** 20%
  - **One Shot Prompting:** 30%
  - **One Shot with Minor Tweaks:** 10%
  - **Few Shot Prompting:** 35%
  - **Few Shot Prompting with Minor Tweaks:** 10%
  - **Hybrid Approach:** 50%
  - **Semantic Approach:** 65%
  - **Short Circuiting:** 50%

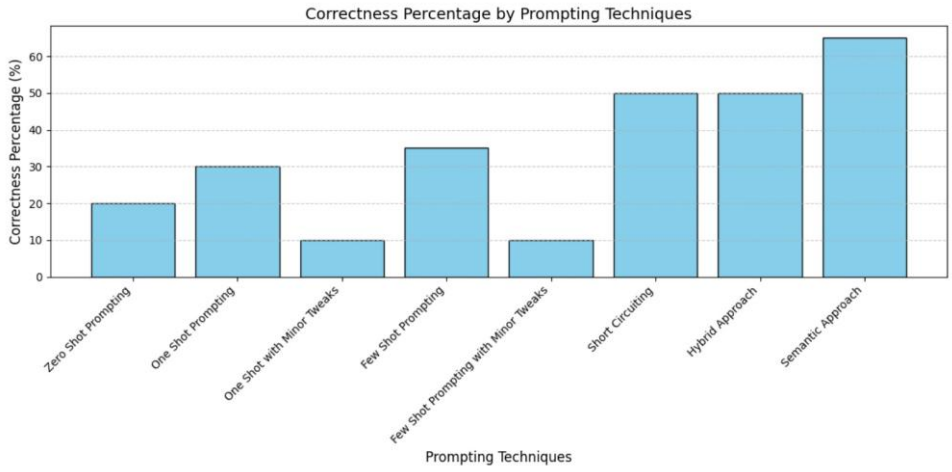

Figure 2: Results after using different Prompting Techniques

Future Work

Although our approach has shown good results, there is still room for improvement. Some areas we plan to explore further include:

- **Fine-Tuning Models:** Training the transformer models on a larger dataset could potentially improve the accuracy when dealing with complex REGEX transformations.
- **Combination of Techniques:** We believe that combining different prompting strategies could lead to better results, especially when dealing with diverse or more complex patterns. This approach could harness the strengths of multiple techniques.
- **Advanced Constructs:** There is a need to develop more specialized methods to handle advanced REGEX features, such as lookaheads or repetition patterns, which could further enhance the framework.
- **Automation:** We are also considering automating the process of semantic validation using parsers or custom-built tools. This would help streamline the workflow and make the process more efficient.

These enhancements will help extend the usability of our approach, making it more reliable and efficient for complex patterns.

Conclusion

Prompting techniques produce varying results depending on the type of REGEX being used. To improve accuracy, a hybrid or adaptive prompting approach can be adopted, tailoring prompts to specific types of REGEX. Fine-tuning the model with a diverse range of REGEX patterns and their corresponding LIKE representations further enhances its ability to generalize. Additionally, incorporating context-specific data, such as REGEX patterns for emails, IP addresses, and other domains, can refine the model's precision for specialized use cases. Transfer learning can also be applied to enable crossdomain REGEX-to-LIKE conversion, where patterns learned in one domain (e.g., email addresses) are leveraged to improve performance in another (e.g., phone numbers), allowing the model to adapt effectively across various scenarios.

References

1 H. Touvron et al: LLaMA: Open and Efficient Foundation Language Models. 10.48550/arXiv.2302.13971

2 Prompt Engineering: Google Developers - Machine Learning Resources: Prompt Engineering

3 Regex Expressions: NTU - Regular Expressions Tutorial