Reliability Testing for LLM-Based Systems

Robert Cunningahm

August 30th 2024

1 Introduction

As large language models (LLMs) become increasingly integrated into various applications, ensuring their reliability is critical. These systems often take multiple inputs and produce corresponding outputs, each of which must adhere to specific guidelines or criteria. Assessing the reliability of such systems is essential for maintaining trust, safety, and effectiveness. This white paper introduces a framework for conducting reliability tests on LLM-based systems. The framework utilizes validators and verifiers to evaluate the system's behavior across multiple dimensions, providing a comprehensive assessment of its performance.

2 Key Concepts in Reliability Testing

2.1 Validators: Ensuring Consistent Behavior

Validators are the foundational elements of the reliability testing framework. They are designed to measure how reliably the system adheres to specific instructions or behaviors. For instance, consider a scenario where an LLM is instructed not to use contractions like "isn't," "doesn't," or "can't." A validator can be implemented to assess how well the model follows this rule.

Example Validator:

```
Validator(
    name = "contraction_validator",
    message = "Output contains too many contractions",
    predicate = lambda o: sum([1 for i in o if "contraction in" in i]) <= 3,
    minimum_success_percentage = 0.95
)
```

Each validator operates on a collection of outputs generated by the system, determining a success percentage that reflects the proportion of outputs meeting the specified criterion. Sometimes validators can have conditional predicates that rely on the input as well:

Conditional Validator:

```
Validator(
   name = "politeness_validator",
   message = "System seems to have forgotten its manners",
   predicate = lambda i, o:
      1 if "Thank you" in i and "You're welcome" in o else
      1 if "Thank you" not in i else
      0,
   success_percentage = 0.90
)
```

2.2 Running Binomial Experiments with Validators

Binomial experiments are used to quantify the reliability of the system as determined by a validator. In a Continuous Alignment Testing (CAT) environment, each validator has a minimum success percentage threshold. The outcome of the binomial experiment is compared against this threshold to determine if the system's behavior is reliable. There are two primary methods for conducting these experiments that can be combined into a third.

2.2.1 Varying Inputs, Single Output

The system generates a single output for each varied input, and the validator assesses the entire collection of outputs.

$$\begin{aligned}&\text{input}_1 \to \text{output}_{11}\\&\text{input}_2 \to \text{output}_{12}\\&\vdots\\&\text{input}_N \to \text{output}_{1N}\end{aligned}$$

$$\text{Validator} : (\text{output}_{11}, \text{output}_{12}, \ldots, \text{output}_{1N}) \to \text{“success percentage”}$$

2.2.2 Fixed Input, Multiple Outputs

A single input is used to generate multiple outputs, and the validator assesses this set of outputs.

$$\text{input}_J \to \text{output}_{1J}, \text{output}_{2J}, \text{output}_{3J}, \ldots, \text{output}_{NJ}$$

$$\text{Validator} : (\text{output}_{1J}, \text{output}_{2J}, \ldots, \text{output}_{NJ}) \to \text{“success percentage”}$$

2.2.3 Varying Inputs, Multiple Outputs

In this scenario, N^2 outputs are generated. Given the time required to produce these outputs, it is crucial to optimize validation strategies to make the most of them. Since each validation from the validator is linearly independent, combining the "success percentages" should be done thoughtfully.

$$\begin{aligned}
&\text{input}_{11} \to \text{output}_{11}, \text{output}_{21}, \text{output}_{31}, \ldots, \text{output}_{N1}\\
&\text{input}_{12} \to \text{output}_{12}, \text{output}_{22}, \text{output}_{32}, \ldots, \text{output}_{N2}\\
&\vdots\\
&\text{input}_{1N} \to \text{output}_{1N}, \text{output}_{2N}, \text{output}_{3N}, \ldots, \text{output}_{NN}
\end{aligned}$$

$$\text{Validator} : (\text{appropriate subset of outputs}) \to \text{“success percentage”}$$

Validation Strategies:

- Validating rows
- Validating columns
- Validating all N^2 outputs

3 Scaling Reliability Testing with Multiple Validators

3.1 Understanding the Role of Multiple Validators

In real-world applications, it is often necessary to assess the reliability of a system across multiple dimensions simultaneously. This requires deploying multiple validators, each designed to measure a specific aspect of the system's behavior. In this section, we extend the framework discussed earlier to accommodate the use of K validators. This approach allows for a more comprehensive evaluation of the system's reliability, as it accounts for the diverse requirements and constraints that a system may need to satisfy.

Example Use Case:

Consider a content generation system where the LLM must adhere to the following rules:

1. No Contractions: Avoid using contractions in the output.
2. Factual Accuracy: Ensure that all statements are factually correct.
3. Ethical Compliance: Avoid generating content that could be considered biased or offensive.
4. Tone Consistency: Maintain a consistent, professional tone throughout the output.

Each of these rules would be represented by a separate validator:

```
Validator(
    name = "contraction_validator",
    message = "Output contains too many contractions",
    predicate = lambda o: sum([1 for i in range(N) if "contraction in" in o[i]])
         <= 3,
    minimum_success_percentage = 0.95
)

Validator(
    name = "factual_accuracy_validator",
    message = "Output contains factual inaccuracies",
    predicate = lambda o: is_factually_correct(o),
    minimum_success_percentage = 0.98
)

Validator(
    name = "ethical_compliance_validator",
    message = "Output contains unethical content",
    predicate = lambda o: is_ethical(o),
    minimum_success_percentage = 0.99
)

Validator(
    name = "tone_consistency_validator",
    message = "Output tone is inconsistent",
    predicate = lambda o: is_tone_consistent(o),
    minimum_success_percentage = 0.97
)
```

3.2 Running Binomial Experiments with Multiple Validators

When running binomial experiments with multiple validators, the process can be scaled to evaluate the system's output against each validator independently. The success percentage for each validator is computed based on how well the outputs satisfy the corresponding criterion. The overall reliability of the system is then assessed by combining the results of all validators.

3.2.1 Method 1: Varying Inputs with Multiple Validators

In this method, we vary the inputs, generate outputs for each input, and then apply all K validators to the resulting set of outputs. This approach allows us to assess the system's performance across different scenarios.

$$\begin{aligned}
&\text{input}_1 \rightarrow \text{output}_{11}\\
&\text{input}_2 \rightarrow \text{output}_{12}\\
&\vdots\\
&\text{input}_N \rightarrow \text{output}_{1N}
\end{aligned}$$

$$\begin{aligned}
&\text{Validator}_1 : (\text{output}_{11}, \text{output}_{12}, \ldots, \text{output}_{1N}) \rightarrow \text{“success percentage”}_1\\
&\text{Validator}_2 : (\text{output}_{11}, \text{output}_{12}, \ldots, \text{output}_{1N}) \rightarrow \text{“success percentage”}_2\\
&\vdots\\
&\text{Validator}_K : (\text{output}_{11}, \text{output}_{12}, \ldots, \text{output}_{1N}) \rightarrow \text{“success percentage”}_K
\end{aligned}$$

3.2.2 Method 2: Fixed Input with Multiple Validators

In this method, we hold a single input constant and generate multiple outputs for that input. Each validator is then applied to the set of outputs. This method is particularly useful for assessing the consistency of the system's behavior when responding to a single prompt.

$$\text{input}_K \rightarrow \text{output}_{1K}, \text{output}_{2K}, \text{output}_{3K}, \ldots, \text{output}_{NK}$$

$$\begin{aligned}
&\text{Validator}_1 : (\text{output}_{1K}, \text{output}_{2K}, \ldots, \text{output}_{NK}) \rightarrow \text{“success percentage”}_1 \\
&\text{Validator}_2 : (\text{output}_{1K}, \text{output}_{2K}, \ldots, \text{output}_{NK}) \rightarrow \text{“success percentage”}_2 \\
&\vdots \\
&\text{Validator}_K : (\text{output}_{1K}, \text{output}_{2K}, \ldots, \text{output}_{NK}) \rightarrow \text{“success percentage”}_K
\end{aligned}$$

3.3 Aggregating Results from Multiple Validators

After obtaining the success percentages from all K validators, the next step is to aggregate these results to form a comprehensive view of the system's reliability. There are several ways to approach this aggregation, depending on the specific requirements of the system and the relative importance of each validator.

3.3.1 Simple Average Method

One straightforward approach is to calculate the simple average of the success percentages across all validators. This method treats each validator equally, providing a general measure of the system's overall reliability.

$$\text{Overall Success Percentage} = \frac{1}{K}\sum_{i=1}^{K} \text{Success Percentage}_i$$

3.3.2 Weighted Average Method

In cases where certain behaviors are more critical than others, a weighted average can be used. Each validator is assigned a weight based on its importance, and the overall success percentage is calculated as the weighted sum of the individual success percentages.

$$\text{Overall Success Percentage} = \frac{\sum_{i=1}^{K} \text{Weight}_i \times \text{Success Percentage}_i}{\sum_{i=1}^{K} \text{Weight}_i}$$

3.3.3 Minimum Threshold Method

Another approach is to set a minimum success threshold that the system must meet across all validators. The overall reliability is then determined by the lowest success percentage recorded among the validators. This method is stringent, ensuring that the system performs reliably across all critical dimensions.

$$\text{Overall Success Percentage} = \min \begin{pmatrix} \text{Success Percentage}_1, \\ \text{Success Percentage}_2, \\ \vdots \\ \text{Success Percentage}_K \end{pmatrix}$$

3.4 Confidence Intervals with Multiple Validators

Confidence intervals provide a range within which the true success percentage likely falls. When dealing with multiple validators, confidence intervals can be calculated for each validator's success percentage. These intervals can then be reported individually or combined to provide a more nuanced understanding of the system's reliability.

For each validator, the confidence interval is calculated using the formula:

$$\text{Confidence Interval for Validator } i = \text{Success Percentage}_i \pm Z \times \text{SE}_i$$

Where SE_i is the standard error for validator i, calculated as:

$$\text{SE}_i = \sqrt{\frac{\text{Success Percentage}_i \times (1 - \text{Success Percentage}_i)}{N_i}}$$

The combined confidence interval for the system's overall reliability can be determined based on the method of aggregation used.

3.5 Parallel Execution of Validators

One of the key advantages of using multiple validators is that they can be executed in parallel. This parallelism allows for efficient and scalable testing, particularly in Continuous Alignment Testing (CAT) environments where real-time feedback is crucial.

By running multiple validators simultaneously, the system can quickly identify areas where it meets or falls short of expectations, enabling prompt adjustments and improvements.

4 Verifiers: Assessing System-Wide Reliability

Verifiers provide a holistic assessment of the system's reliability. Unlike validators, which focus on specific aspects of behavior that can be evaluated programmatically, verifiers evaluate the overall performance of the system. A verifier reviews input-output pairs and determines whether the system's output passes or fails based on a comprehensive set of instructions.

Verifier Process:

$$\begin{aligned} &\text{input}_1 \rightarrow \text{output}_1 \\ &\text{input}_2 \rightarrow \text{output}_2 \\ &\vdots \\ &\text{input}_N \rightarrow \text{output}_N \end{aligned}$$

$$\begin{aligned}
&\text{Verifier} : (\text{input}_1, \text{output}_1) \rightarrow \text{PASS/FAIL}\\
&\text{Verifier} : (\text{input}_2, \text{output}_2) \rightarrow \text{PASS/FAIL}\\
&\vdots\\
&\text{Verifier} : (\text{input}_N, \text{output}_N) \rightarrow \text{PASS/FAIL}
\end{aligned}$$

The results of these verification steps are aggregated to assess the overall reliability of the system, but the addition of another call to an LLM in this verification step opens up the possibility for the system to self-correct on a single input-output pair basis.

4.1 Verifier Driven Retry Mechanism

Once you have integrated AI into your application, there are several ways to make the system auto-correct. One of the simplest is to use the verification step to trigger a "retry."

Since the verifier step uses an LLM transaction to decide whether or not the input-output pair "passes" our test, that same LLM can be made to provide a list of reasons for the failure. In this case, the input can be augmented with those reasons and sent back through the system to produce another output for the system. This can be made to repeat MAX times.

$$\begin{aligned}
&(\text{input}, \text{output}_1) \rightarrow \text{Verifier: PASS}\\
&\quad \#\ \text{publish output}_1
\end{aligned}$$

$$\begin{aligned}
&(\text{input}, \text{output}_1) \rightarrow \text{Verifier: FAIL, reasons}\\
&\quad \rightarrow (\text{input} + \text{reasons}, \text{output}_2) \rightarrow \text{Verifier: PASS}\\
&\quad \#\ \text{publish output}_2
\end{aligned}$$

$$\begin{aligned}
&(\text{input}, \text{output}_1) \rightarrow \text{Verifier: FAIL, reasons}_1\\
&\quad \rightarrow (\text{input} + \text{reasons}_1, \text{output}_2) \rightarrow \text{Verifier: FAIL, reasons}_2\\
&\quad \rightarrow (\text{input} + \text{reasons}_2, \text{output}_3) \rightarrow \text{Verifier: FAIL, reasons}_3\\
&\quad \vdots\\
&\quad \rightarrow (\text{input} + \text{reasons}(\text{MAX} - 1), \text{output}_{\text{MAX}}) \rightarrow \text{Verifier: FAIL, reasons}_{\text{MAX}}\\
&\quad \#\ \text{no output published}
\end{aligned}$$