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: o.count("'") <= 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:
        "You're welcome" in o"You're welcome" in o
        if "Thank you" in i
        else True,
    minimum_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.

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

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

2.2.2 Fixed Input, Multiple Outputs

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

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

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

2.2.3 Varying Inputs, Multiple Outputs

In this approach, the system generates multiple outputs for each varied input, resulting in a comprehensive set of N^2 outputs. This method enables a thorough examination of the system's behavior across a wide range of scenarios, capturing both the variability in inputs and the stochastic nature of output generation.

$$\begin{aligned}
&\text{input}_1 \rightarrow \text{output}_{11}, \text{output}_{12}, \text{output}_{13}, \ldots, \text{output}_{1N} \\
&\text{input}_2 \rightarrow \text{output}_{21}, \text{output}_{22}, \text{output}_{23}, \ldots, \text{output}_{2N} \\
&\text{input}_3 \rightarrow \text{output}_{31}, \text{output}_{32}, \text{output}_{33}, \ldots, \text{output}_{3N} \\
&\quad\vdots \\
&\text{input}_N \rightarrow \text{output}_{N1}, \text{output}_{N2}, \text{output}_{N3}, \ldots, \text{output}_{NN}
\end{aligned}$$

Due to the significant number of outputs, an effective validation strategy is crucial to efficiently assess the system's reliability. There are three primary methods for applying validators in this context:

Validating Rows Validating rows involves assessing all outputs generated from a single input. For each input i, the validator is applied to the set of outputs $\{\text{output}_{i1}, \text{output}_{i2}, \ldots, \text{output}_{iN}\}$.

$$\begin{aligned}
\text{Validator for input } i : (\text{output}_{i1}, \text{output}_{i2}, \ldots, \text{output}_{iN}) \\
\rightarrow \text{“Success Percentage for Input } i\text{”}
\end{aligned}$$

This method evaluates the system's consistency and reliability in handling a specific input across multiple output variations. It helps identify inputs for which the system consistently performs well or poorly, highlighting potential input-specific issues.

Validating Columns Validating columns focuses on outputs generated across different inputs under the same conditions or iterations. For each output index j, the validator is applied to the set $\{\text{output}_{1j}, \text{output}_{2j}, \ldots, \text{output}_{Nj}\}$.

$$\begin{aligned}\text{Validator for iteration } j : (&\text{output}_{1j}, \text{output}_{2j}, \ldots, \text{output}_{Nj}) \\ &\rightarrow \text{“Success Percentage for Iteration } j\text{”}\end{aligned}$$

This approach assesses the system's performance across a variety of inputs for a particular output generation instance. It can reveal systemic issues that affect all inputs under certain generation conditions, such as biases introduced by specific random seeds or sampling methods.

Validating All N^2 Outputs Validating all N^2 outputs involves applying the validator to every output individually and aggregating the results.

$$\begin{aligned}\text{Validator} : (&\text{output}_{11}, \text{output}_{12}, \ldots, \text{output}_{NN}) \\ &\rightarrow \text{“Overall Success Percentage”}\end{aligned}$$

This comprehensive method provides a holistic view of the system's reliability across all inputs and outputs. While thorough, it may be resource-intensive, necessitating strategies like parallel processing or intelligent sampling to remain practical.

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, each represented by a corresponding validator:

```
Validator(
    name = "contraction_validator",
    message = "Output contains too many contractions",
    predicate = lambda o: o.count("'") <= 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
)
```

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.

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.

3.6 Generalizing to a Tensor Framework for Reliability Analysis

When extending reliability testing to include multiple inputs, multiple outputs per input, and multiple validators, the system's performance can be represented as a three-dimensional tensor. This **Reliability Tensor** captures the interplay between inputs, outputs, and validators, allowing for a nuanced analysis of the system's reliability.

3.6.1 Constructing the Reliability Tensor

The Reliability Tensor R can be defined with dimensions corresponding to:

- **Input Dimension (I):** Represents the set of varied inputs: $\{\text{input}_1, \text{input}_2, \ldots, \text{input}_N\}$.
- **Output Dimension (J):** Represents the multiple outputs generated per input: $\{\text{output}_1, \text{output}_2, \ldots, \text{output}_M\}$.

- **Validator Dimension (K):** Represents the set of validators: $\{\text{validator}_1, \text{validator}_2, \ldots, \text{validator}_K\}$.

Each element $R[i][j][k]$ in the tensor represents the result (e.g., pass/fail, success percentage) of validator k applied to output output_j generated from input input_i.

$$R[i][j][k] = \text{Result of validator } k \text{ on output}_j \text{ from input}_i$$

3.6.2 Analyzing Success Percentages Along Tensor Axes

By examining the tensor along different axes, we can derive various success percentages:

- **Per-Input Success Rates:** For each input i, aggregate results across outputs and validators to assess how reliably the system handles that specific input.

$$\text{Success Percentage for Input } i = \text{Aggregate}_{j,k}\, R[i][j][k]$$

- **Per-Output Success Rates:** For each output iteration j, aggregate results across inputs and validators to evaluate the reliability of outputs generated under specific conditions.

$$\text{Success Percentage for Output } j = \text{Aggregate}_{i,k}\, R[i][j][k]$$

- **Per-Validator Success Rates:** For each validator k, aggregate results across inputs and outputs to measure how well the system performs regarding a specific criterion.

$$\text{Success Percentage for Validator } k = \text{Aggregate}_{i,j}\, R[i][j][k]$$

3.6.3 Developing Terms of Art

To facilitate discussion and analysis, we introduce the following terms:

- **Input Reliability Profile (IRP):** The collection of success percentages for a specific input across all outputs and validators.
- **Output Reliability Profile (ORP):** The collection of success percentages for a specific output iteration across all inputs and validators.
- **Validator Reliability Profile (VRP):** The collection of success percentages for a specific validator across all inputs and outputs.

These profiles help identify patterns and anomalies in the system's performance, enabling targeted improvements.

3.6.4 Marginal Success Percentages and Reliability Profiles

By aggregating over specific dimensions of the tensor, we can compute marginal success percentages that provide insights into different aspects of system performance.

- **Input Marginal Success Percentage (Input MSP):** The success percentage for each input i, aggregated over outputs and validators.

$$\text{Input MSP}[i] = \frac{1}{J \times K} \sum_{j,k} R[i][j][k]$$

- **Output Marginal Success Percentage (Output MSP):** The success percentage for each output iteration j, aggregated over inputs and validators.

$$\text{Output MSP}[j] = \frac{1}{I \times K} \sum_{i,k} R[i][j][k]$$

- **Validator Marginal Success Percentage (Validator MSP):** The success percentage for each validator k, aggregated over inputs and outputs.

$$\text{Validator MSP}[k] = \frac{1}{I \times J} \sum_{i,j} R[i][j][k]$$

These marginal success percentages form the basis of the Input Reliability Profile (IRP), Output Reliability Profile (ORP), and Validator Reliability Profile (VRP), respectively.

3.6.5 Interpreting Reliability Profiles

- **Input Reliability Profile (IRP):** Highlights inputs where the system performs exceptionally well or poorly, guiding efforts to improve handling of specific inputs.
- **Output Reliability Profile (ORP):** Reveals output iterations that consistently yield better or worse results, potentially indicating issues with certain generation methods or configurations.
- **Validator Reliability Profile (VRP):** Indicates areas where the system meets or fails to meet specific criteria, informing adjustments to enhance compliance with critical requirements.

3.6.6 Visualizing the Reliability Tensor

To aid in interpreting the data, visualization techniques such as heatmaps or 3D plots can represent the tensor's elements and marginal percentages. Such visualizations can make patterns and outliers more apparent, facilitating a deeper understanding of the system's performance.

3.6.7 Framework for Combining Success Percentages

To report an overall reliability score for the system, we can aggregate success percentages from the tensor using various methods:

- **Mean Aggregation:** Compute the average success percentage across all elements.

$$\text{Overall Success Percentage} = \frac{1}{I \times J \times K} \sum_{i,j,k} R[i][j][k]$$

- **Weighted Aggregation:** Assign weights to inputs, outputs, or validators based on their importance.

$$\text{Overall Success Percentage} = \frac{\sum_{i,j,k} W[i][j][k] \times R[i][j][k]}{\sum_{i,j,k} W[i][j][k]}$$

- **Minimum Threshold Method:** Identify the lowest success percentage across any dimension to ensure reliability standards are met in all areas.

$$\text{Overall Success Percentage} = \min_{i,j,k} R[i][j][k]$$

The choice of aggregation method depends on the specific requirements and priorities of the system being evaluated.

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:

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

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.

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

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

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

4.2 Retry Mechanisms with Validators

In complex LLM-based systems, outputs may occasionally fail to meet all the criteria specified by multiple validators due to the inherent stochasticity of language models. To enhance reliability, a **retry mechanism** can be implemented, allowing the system to generate new outputs for a given input up to a maximum of m attempts. This section explores how to design such a retry mechanism using all validators and how to predict the expected number of retries required for an output to pass all validators based on their success percentages.

4.2.1 Concept of the Retry Mechanism

The retry mechanism operates as follows:

1. **Initial Generation:** For a given input i, the system generates an output o_1.
2. **Validation:** All validators $\{V_1, V_2, \ldots, V_K\}$ are applied to o_1.
3. **Check Pass/Fail:**
   - If o_1 passes all validators, the process stops, and o_1 is accepted.
   - If o_1 fails any validator, the system retries up to a maximum of m times.
4. **Subsequent Generations:** On each retry j, the system generates a new output o_j for the same input i and repeats the validation process.
5. **Termination Conditions:**
   - **Success:** If any o_j passes all validators before reaching m retries, the output is accepted.
   - **Failure:** If none of the outputs pass all validators after m retries, the process terminates without an accepted output.

4.2.2 Predicting the Expected Number of Retries

To optimize the retry mechanism, it is crucial to predict:

- The expected number of retries needed for an output to pass all validators.
- The optimal value of m to balance reliability and resource consumption.

Success Percentages of Validators Each validator V_k has an inherent success percentage p_k, representing the probability that a randomly generated output will pass V_k.

- **Validator Success Probability:** p_k = Success Percentage of V_k
- **Assumption:** The validators operate independently, and the success probabilities are consistent across outputs for a given input.

Combined Success Probability The probability that an output passes all validators is:

$$P_{\text{pass}} = \prod_{k=1}^{K} p_k$$

This formula assumes independence among validators.

Expected Number of Retries The expected number of retries $E[R]$ required for an output to pass all validators is:

- **Geometric Distribution:** Since each attempt is independent, and the probability of success remains constant, the number of trials until the first success follows a geometric distribution.
- **Expected Number of Trials (including the first attempt):**

$$E[R] = \frac{1}{P_{\text{pass}}}$$

- **Expected Number of Retries (excluding the first attempt):**

$$E[\text{Retries}] = E[R] - 1 = \frac{1}{P_{\text{pass}}} - 1$$

Determining Maximum Retries m To choose an appropriate maximum number of retries m:

- **Probability of Success Within m Attempts:**

$$P_{\text{success within } m \text{ attempts}} = 1 - (1 - P_{\text{pass}})^m$$

- **Selecting m:** Choose m such that $P_{\text{success within } m \text{ attempts}}$ meets a desired confidence level (e.g., 95%).

4.2.3 Is m Input-Specific?

The value of m can be:

- **Input-Agnostic:** If the success probabilities p_k are consistent across all inputs, m can be set globally.
- **Input-Specific:** If success probabilities vary significantly with different inputs (as indicated by the Input Reliability Profile), m may need adjustment per input.

Using the Reliability Tensor The Reliability Tensor $R[i][j][k]$ provides empirical success data for each input i, output attempt j, and validator k.

- **Empirical Success Probability for Input i:**

$$P_{\text{pass},i} = \frac{1}{J} \sum_{j=1}^{J} \left(\prod_{k=1}^{K} R[i][j][k] \right)$$

- **Expected Retries for Input i:**

$$E[R]_i = \frac{1}{P_{\text{pass},i}}$$

If $P_{\text{pass},i}$ varies significantly across inputs, it indicates that m should be adjusted per input to optimize performance.

4.2.4 Practical Calculation Example

Assumptions:

- Validators and their success percentages:
  - V_1: $p_1 = 0.95$
  - V_2: $p_2 = 0.90$
  - V_3: $p_3 = 0.85$
- **Combined Success Probability:**

$$P_{\text{pass}} = p_1 \times p_2 \times p_3 = 0.95 \times 0.90 \times 0.85 = 0.72675$$

- **Expected Number of Trials:**

$$E[R] = \frac{1}{0.72675} \approx 1.376$$

- **Expected Number of Retries:**

$$E[\text{Retries}] = E[R] - 1 \approx 0.376$$

- **Conclusion:** On average, less than one retry is needed for an output to pass all validators.

Determining m for 99% Confidence:

- Desired $P_{\text{success within } m \text{ attempts}} = 0.99$
- Solve for m:

$$\begin{aligned} 0.99 &= 1 - (1 - 0.72675)^m \\ (1 - 0.72675)^m &= 0.01 \\ (0.27325)^m &= 0.01 \\ m \log(0.27325) &= \log(0.01) \\ m &= \frac{\log(0.01)}{\log(0.27325)} \approx 2.74 \end{aligned}$$

- **Conclusion:** Set $m = 3$ retries to have a 99% chance of success.

4.2.5 Factors Influencing m

Validator Independence

- **Assumption of Independence:** The calculation assumes validators act independently.
- **Correlation Between Validators:** If validators are correlated, the combined success probability may differ, affecting m.

Input Variability

- **Input-Specific Success Rates:** Use the Reliability Tensor to identify inputs with lower success probabilities.
- **Adaptive Retry Mechanism:** Adjust m based on input-specific data to optimize resource usage.

System Constraints

- **Resource Limitations:** Higher m increases computational load and latency.
- **User Experience:** Excessive retries may delay responses; balance is necessary.

4.2.6 Implementing the Retry Mechanism

Algorithm Steps:

1. **Initialize:** Set maximum retries m, initialize attempt counter $j = 1$.
2. **Generate Output:** Produce output o_j for input i.
3. **Validation:** Apply all validators V_k to o_j.
4. **Check Pass/Fail:**
   - **If Pass:** Accept o_j, terminate.
   - **If Fail:** Increment j.
5. **Retry Condition:**
   - **If** $j \leq m$: Go back to Step 2.
   - **If** $j > m$: Fail the input, terminate.

Considerations:

- **Logging:** Record each attempt and validation results for analysis.
- **Timeouts:** Implement time constraints to prevent indefinite processing.
- **Feedback Loop:** Analyze failed inputs to improve model or validators.