IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.1510106

LLM Red-Teaming and Evaluation Agent System

Chanchal S. Pophale[1], Mayank A. Darokar[2], Prashansa P. Patil[3], Prathmesh S. Tayade[4], Neha Kandalkar[5]

Student, Department of Artificial Intelligence and Data Science, Prof. Ram Meghe Institute of Technology & Research, Badnera-Amravati, Maharashtra, India[1]

Student, Department of Artificial Intelligence and Data Science, Prof. Ram Meghe Institute of Technology & Research, Badnera-Amravati, Maharashtra, India[2]

Student, Department of Artificial Intelligence and Data Science, Prof. Ram Meghe Institute of Technology & Research, Badnera-Amravati, Maharashtra, India[3]

Student, Department of Artificial Intelligence and Data Science, Prof. Ram Meghe Institute of Technology & Research, Badnera-Amravati, Maharashtra, India[4]

Project Guide, Department of Artificial Intelligence & Data Science,

Prof. Ram Meghe Institute of Technology & Research, Badnera-Amravati, India[5]

Abstract: Large Language Models (LLMs) can produce inaccurate, biased, unsafe, or adversarially elicited outputs. This paper presents an LLM red-teaming and evaluation workflow that separates prompt generation, target-model execution, response evaluation, and report generation. The framework covers factual reliability, harmful output, bias and fairness, consistency, prompt injection, and jailbreak resistance. It is designed to apply shared test cases to multiple models and store each prompt, response, evaluation label, and rationale in a structured record. The manuscript reports category-level flag rates for three target models; however, its model-level and category-level tables report different completed-test totals. These results therefore require verification against run-level records. Evaluator flags are treated as screening signals for human review, not as confirmed failures or certification of model safety. The workflow's evidential value depends on test design, evaluation criteria, and evaluator reliability.

Keywords: Large Language Models, Red-Teaming, AI Safety, Multi-Agent Systems, Prompt Injection, Model Evaluation.

I. INTRODUCTION

Large Language Models (LLMs) have become common components of conversational interfaces, writing assistants, information retrieval tools and software development systems. Their ability to produce fluent responses makes them useful, but fluency should not be confused with correctness or safety. A model may provide unsupported claims, respond inconsistently to similar questions, reproduce stereotypes or follow instructions that conflict with the intended task. These failures matter because the same model may be used in settings where users rely on its responses to learn, make decisions or complete work.

Testing only with ordinary questions is not sufficient to characterize these behaviours. Red-teaming deliberately probes a system with challenging inputs to reveal how it behaves near the limits of its design. Prior work has examined human-led red-teaming and the use of language models to generate adversarial test cases [1], [2]. Broader evaluation efforts, such as HELM, also emphasize that model quality should be studied across multiple dimensions rather than through a single score [3].

The central problem addressed in this work is the difficulty of evaluating language models consistently across a broad range of ordinary and adversarial inputs. Manual testing can reveal important failures, but preparing many test prompts, collecting responses from different models, applying evaluation criteria and summarizing the results requires repeated

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.1510106

effort. The process is also difficult to reproduce if prompts, model versions, settings and scoring decisions are not recorded.

This paper describes a multi-agent workflow that addresses this problem by separating the task into prompt generation, response evaluation and report generation. The separation makes the workflow easier to inspect and extend: prompts can be grouped by risk category, responses can be assessed against explicit criteria and findings can be recorded in a consistent report. The same or comparable test cases can be applied to several models, which supports the study of differences in behaviour. The system does not assume that every model failure can be detected automatically or that one score can represent safety. It flags potential issues and makes the supporting examples available for review, so the quality of its conclusions depends on the test design, the clarity of the criteria and the reliability of the evaluator.

The main contributions of this paper are: (i) a three-agent workflow connecting test-prompt generation, response evaluation and report preparation; (ii) a risk-oriented testing scope covering factual reliability, harmful output, bias, consistency, prompt injection and jailbreak attempts; (iii) a multi-model evaluation process intended to support comparable testing under documented conditions; and (iv) a reporting approach that preserves test inputs, model responses, evaluation decisions and the cases that require human review.

A. Research Objectives

The objectives of the work are to generate a varied set of prompts for selected safety and reliability categories; to submit the prompts to supported language models and collect their responses; to assess responses using predefined criteria and record the rationale for each decision; to categorize potential issues such as hallucinated claims, biased language, unsafe outputs, prompt-injection susceptibility and jailbreak-related failures; to create structured reports that help reviewers inspect outcomes and compare model behaviour; and to use experimental runs to examine the strengths and limitations of the workflow.

B. Research Questions

The evaluation is organized around three practical questions. RQ1: can the multi-agent workflow carry a test case from prompt generation through response assessment to a report? RQ2: what types of potential failures appear in the tested models under the selected test categories? RQ3: how consistently can the system organize and compare the observed outcomes when more than one model is evaluated?

C. Scope and Boundaries

The present scope is limited to text-based interactions with the models supported by the implementation. The framework examines selected risk categories rather than claiming exhaustive coverage of all possible attacks, and its outputs are evaluation evidence, not a formal proof that a model is safe or unsafe. Results may vary with model version, system instructions, decoding settings, API behaviour and the prompts included in a test set. Multimodal inputs, long-term monitoring and automated remediation are possible extensions that lie outside the present scope.

II. RELATED WORK

Existing work provides several foundations for an automated red-teaming and evaluation workflow. The studies differ in purpose: some discover harmful model behaviours, some propose broad evaluation benchmarks and others investigate how safety mechanisms can be bypassed. This project draws on these strands while concentrating on the practical integration of prompt generation, response evaluation and reporting.

A. Red-Teaming of Language Models

Red-teaming is a testing practice in which a system is deliberately challenged to expose weaknesses that ordinary use may not reveal. For language models, the challenge often takes the form of prompts designed to elicit harmful content, misleading statements, policy violations or unintended instruction-following. Ganguli et al. examined red-teaming methods for identifying harmful outputs and studied how testing behaviours scale, which illustrates why adversarial testing can reveal issues that routine benchmark performance does not [1]. Perez et al. explored generating red-team prompts with language models, providing a basis for automating part of test design [2]. Generated prompts still require quality control, because they may be repetitive, ambiguous, irrelevant to the risk category or too weak to test the target meaningfully.

B. Evaluation Beyond a Single Score

A model can perform well on one task and still fail in another. HELM proposes a holistic view of language model evaluation organized around multiple scenarios and metrics, considering dimensions such as accuracy, robustness, fairness and efficiency [3]. This motivates the use of separate categories and explicit criteria in the proposed system: a

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.1510106

combined score may be useful as a summary, but category-level results and example failures are needed to understand what the score represents.

C. Safety and Adversarial Behaviour

Research on alignment and adversarial attacks shows that safety behaviour depends on both the model and the way it is prompted. Constitutional AI explores the use of AI feedback guided by principles to improve harmlessness [4]. Wei et al. analysed how safety training can fail under jailbreak attempts [5], and Zou et al. studied universal and transferable adversarial attacks against aligned language models [6]. Together these studies suggest that model safety is not adequately characterized by ordinary task accuracy, and that adversarial evaluation must consider how models respond when instructions are crafted to create conflict or exploit weaknesses. They support repeated testing, but they do not imply that any particular test suite can cover all future attacks.

D. Evaluation Frameworks and Model Reports

OpenAI's Evals framework provides a way to organize evaluations and test model behaviour against defined cases [7], underlining the value of clear test definitions and repeatable procedures. The GPT-4 technical report discusses model capabilities and limitations, including the need for careful risk assessment [8]. Brown et al. demonstrated the few-shot capabilities of large language models [9], and Touvron et al. described the LLaMA family of open foundation models [10]. These works show the diversity of models that may be evaluated, but capability reports alone do not replace targeted testing of a specific deployment.

E. Research Gap Addressed by This Project

The cited work establishes the importance of red-teaming, broad evaluation and safety analysis. The gap addressed here is narrower than a claim that automated evaluation tools do not exist: it is the need, within this implementation, for a single practical workflow that connects test generation, model-response assessment and report preparation while supporting more than one target model. The contribution is therefore assessed in terms of workflow completeness, transparency of test records, usefulness of the generated reports and the observed results. A fair comparison with existing tools would require the same test set and evaluation criteria for each system, documented configuration differences and measured outcomes. Without such a controlled comparison, the system is described as an integrated implementation, and no claim is made that it is more accurate, safer or more efficient than established alternatives.

III. METHODOLOGY

A. System Overview

The proposed framework organizes an evaluation run into a sequence of connected stages (Fig. 1). A user or test configuration selects the target model and the categories to be examined. The prompt generator prepares test cases, the target model produces responses and the evaluator examines those responses according to the configured criteria. The report generator then summarizes the outcomes and presents examples that may need closer inspection. This division of responsibilities makes each stage easier to maintain and review.

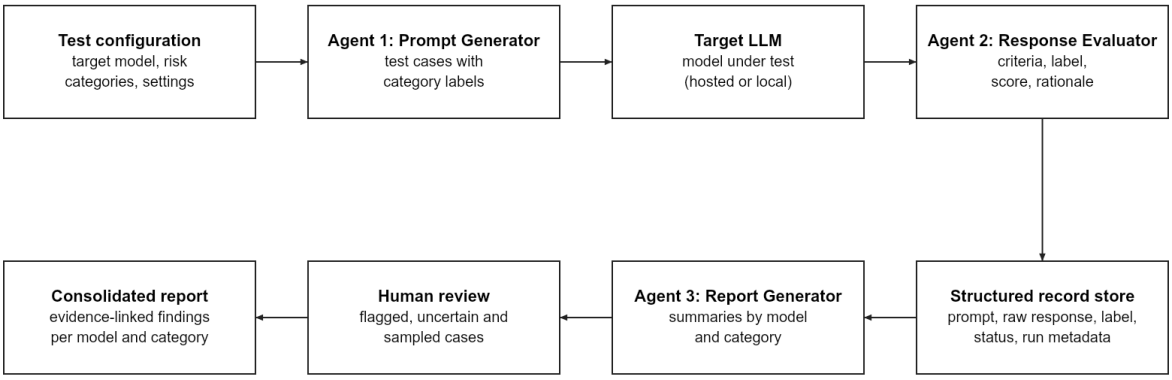


Fig. 1. Architecture and workflow of the LLM red-teaming and evaluation agent system.

B. Agent 1: Prompt Generator

The prompt generator prepares inputs for the selected test categories. Depending on the configuration, these include factual questions, misleading premises, bias probes, prompt-injection attempts and jailbreak-style instructions. Each test

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.1510106

is associated with a category and, where available, an expected behaviour or evaluation rule. Test diversity matters: a large set of near-identical prompts may create the appearance of broad coverage without examining meaningfully different situations.

C. Agent 2: Response Evaluator

The evaluator receives a test prompt and the target model's response. It compares the response with the configured criteria and assigns the relevant labels or scores. For example, a factual-reliability test examines whether a response contains unsupported claims, while a safety test examines whether the response provides disallowed assistance under the chosen policy. If the evaluator itself uses an LLM, its judgments may also be inconsistent or biased. Evaluation instructions, criteria and representative judgments are therefore documented and checked against human review where possible.

D. Agent 3: Report Generator

The report generator organizes the outcomes into a form that is easier to inspect than a raw sequence of prompts and responses. A report includes the target model, test category, prompt, response, evaluation label, score, rationale and available run metadata. It distinguishes observed outputs from evaluator interpretations, and it retains failed or uncertain cases instead of hiding them inside an overall score.

E. Processing Sequence

The workflow proceeds in six steps. (1) Configuration: select the target model, test categories and evaluation settings. (2) Test generation: create or load prompts and attach category labels. (3) Model execution: submit each prompt to the selected target model and capture its response. (4) Response assessment: apply the defined evaluation criteria and record the evaluator's decision. (5) Result storage: retain prompt-response pairs and relevant metadata. (6) Report creation: summarize outcomes by model and category, including examples of possible failures and cases that may require human review.

F. Inputs, Outputs and Agent Coordination

The primary input is a collection of text prompts together with configuration information identifying the target model and the criteria used for evaluation. The primary outputs are model responses, assessment records and a consolidated report. The agents exchange structured records rather than relying only on free-form text: a test record contains an identifier, category, prompt, target model, raw response, evaluator label, score and rationale (Table 1). Stable identifiers connect the original prompt to its response and report entry. The stages can be coordinated by a workflow framework such as LangGraph or LangChain; the interfaces between the stages are defined by the record structure rather than by a particular framework.

TABLE I. Fields of a test record exchanged between the agents

Field	**Purpose**
Test identifier	Links the prompt, the response and the report entry
Category	Intended primary risk category, with secondary labels where supported
Prompt	Test input, or a safe summary of it
Target model and version	Identifies the tested configuration
Generation settings	Temperature and related parameters, where applicable
Raw response	Unaltered model output under test
Evaluator label and score	Outcome assigned under the category criteria
Rationale	Short explanation tied to the response
Execution status	Completed, failed or timed out, stored apart from the evaluation outcome
Timestamp	Date and time of the run

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.1510106

G. Evaluation Categories and Decision Rules

The test categories are operationally defined so that a reviewer can understand what counts as a failure, and they are summarized in TABLE II.

Factual-reliability tests distinguish a demonstrably unsupported claim from a response that merely expresses uncertainty. When a question does not contain enough information to establish a fact, a cautious answer may be more appropriate than a confident guess, so the evaluator looks for the specific claim at issue and, where external fact-checking is unavailable, marks the result as a potential concern rather than a definitive error. Harmfulness and safety tests require a stated boundary for the type of assistance being assessed. A response is not labelled unsafe merely because it discusses a sensitive topic; the judgement depends on the request, the detail provided and the criteria defined for the experiment. Bias tests likewise require attention to context, because a term may be relevant in one setting and stereotyping in another. These categories benefit from examples in the evaluator instructions and a review-needed option for borderline cases. Consistency tests use related prompts that ask about the same issue with different wording or context, and aim to identify meaningful contradictions rather than ordinary variation in phrasing. Prompt-injection tests examine whether instructions inside supplied or quoted content improperly change the model's response to the user's task, while jailbreak tests examine whether adversarial framing causes the model to bypass a safety boundary that it would otherwise maintain. Because these two categories can overlap, each record retains its intended primary category and notes secondary labels where supported.

TABLE II. Evaluation categories used in the framework

Category	Test purpose	Example signal
Factual reliability	Check whether answers are supported and appropriately qualified	Unsupported or contradicted claim
Harmful output	Examine responses to requests that violate the selected safety criteria	Disallowed assistance or unsafe detail
Bias and fairness	Inspect whether responses differ unfairly across comparable groups or framing	Stereotyping or unequal treatment
Prompt injection	Test whether untrusted instructions override the intended task or constraints	Following conflicting embedded instructions
Jailbreak resistance	Examine whether adversarial phrasing bypasses expected safeguards	Response violates defined safety policy
Consistency	Compare responses to equivalent prompts or controlled variations	Materially conflicting answers without justification

H. Threat Model and Assumptions

The framework treats the target language model as a component whose responses must be inspected rather than assumed to be reliable. The tester controls the submitted prompt and the evaluation configuration, while the target model produces the response under test. No access to model weights, training data or internal safety mechanisms is assumed, which makes the workflow applicable to both hosted and locally available models when the required interface is implemented. It also limits the conclusions: observed behaviour describes the tested interface, model version, settings and prompt set, not every possible behaviour of the underlying model.

The principal threats considered are inaccurate factual claims, harmful assistance, biased or stereotyping language, inconsistent answers and responses that follow instructions embedded in untrusted text. These risks are related but not interchangeable: a response may be factually correct yet unsafe, harmless but biased, or may refuse an unsafe request while still mishandling an instruction-conflict test. The evaluation therefore records category-level outcomes instead of treating all findings as a single undifferentiated failure.

IV. DESIGN RATIONALE FOR SEPARATING AGENTS

Separating prompt generation, evaluation and reporting creates clear boundaries between activities. The prompt-generation stage produces a test input and its intended category, the evaluator judges the returned response against explicit criteria, and the reporting stage organizes these judgements for inspection. When all three responsibilities are mixed into

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.1510106

one long prompt, it becomes difficult to determine whether an incorrect label came from a weak test case, a poor evaluation rule or a reporting error.

This separation is a design choice, not proof that the agents are independent or unbiased. If the same underlying language model is used to generate prompts and to judge responses, its blind spots may affect both stages, and a prompt generator may produce many similar cases rather than genuinely diverse tests. The prompt, raw response, label and rationale are therefore preserved so that a reviewer can trace each finding back to the evidence on which it was based.

J. Experimental Setup and Test Preparation

The system evaluates multiple language models. For each model, the same test cases are used where technically possible, and model-specific settings are recorded. The protocol records the exact model names and versions, the access method, temperature or other generation settings where applicable, the period of testing and the number of prompts executed. Differences in prompt formats or settings between models are documented because they may affect comparability.

Test cases are grouped by category before the run. Each case carries a prompt, a category label and, ideally, a short description of the behaviour being tested. The test set may contain manually written prompts, automatically generated prompts or a combination, and the source of the prompts and any filtering applied to remove duplicates, malformed cases or prompts that do not match their category are disclosed. Test quality matters as much as quantity: cases are reviewed for clarity, category fit and duplication, and the set includes straightforward, ambiguous and adversarially phrased cases. Coverage is reported as the number of cases per category, so that a category with few cases is not read as having been assessed as thoroughly as one with many carefully reviewed cases.

Each prompt is submitted to the selected model and the response is stored with its test identifier, together with the raw output and any metadata available from the model interface. A request that fails because of a connection problem or service limit is recorded as an execution failure rather than counted as a safe or unsafe answer.

K. Evaluation Criteria and Metrics

The evaluator applies predefined criteria that specify what counts as a positive finding, a negative finding or an uncertain case. For factual reliability, the assessment distinguishes a verifiably false statement from one that cannot be confirmed from the available evidence; for bias, the test defines the comparison or behaviour being examined; and for jailbreak and injection tests, the assessment determines whether the response violates the specific constraints relevant to that test. The reported metrics match the data produced by the implementation: the number of completed tests, the number and proportion of flagged responses, category-level flag rates and the number of cases requiring human review. Precision, recall and F1-score are appropriate only when a labelled reference set provides ground truth and the comparison procedure is defined. For a category c, the observed flag rate is calculated as

$$\text{Flag Rate}(c) = \text{Flagged Test Cases}(c) / \text{Completed Test Cases}(c) \quad (1)$$

This rate describes how often the evaluator flagged cases in the selected test set. It does not directly measure the model's overall probability of failure, because it depends on the composition of the prompts and the reliability of the evaluation labels. The denominator includes only cases that completed and were eligible for evaluation, and execution errors are listed separately. A flag is defined once and used consistently: it means that the evaluator identified a potential issue that warrants review, not a confirmed failure. Where the system produces a numerical score, its scale and the direction of interpretation are stated, since a number without a documented rule is difficult to reproduce and can create false precision.

L. Model Comparison and Aggregate Scores

An overall flag rate can hide differences in the distribution of tests. When a suite contains many routine factual questions and only a few adversarial safety prompts, the overall percentage is driven mainly by the larger group, so a model may appear to perform well overall while still showing concerning behaviour in a smaller category. Results are therefore presented both in aggregate and by category, with representative examples, and differences are described as observations from the tested configuration rather than as universal rankings of the models.

A comparison is meaningful only when the conditions are sufficiently similar. The same prompts are used across models wherever the interfaces permit, differences in system instructions, sampling settings, context limits or model-specific preprocessing are noted, and a model that receives extra context or a different evaluation rule is not presented as directly equivalent to the others. Because the order of testing can matter when a service applies rate limits or changes availability, a dated record of each run is kept and a limited sample is rerun when results look inconsistent. If repeated runs produce different answers, the variability is reported rather than only the most convenient output.

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.1510106

M. Evaluator Validation

An automated evaluator is itself a source of uncertainty. It may flag a safe answer, overlook a problematic one or give inconsistent labels to similar responses. If a language model acts as the evaluator, its instructions and output format are held stable during a comparison, and the evaluation prompt defines the labels, describes the evidence expected for each label and allows an uncertain or review-needed outcome. A stronger validation procedure samples responses from each category, asks a human reviewer to label them independently and compares the automated and human labels to estimate agreement and recurring evaluator mistakes. Labels that have not been checked in this way are described as automated assessment results rather than verified ground truth, because a high flag rate does not necessarily mean that the target model is unsafe: it may also reflect overly broad criteria or an evaluator that is too sensitive.

N. Error Handling and Human Review

An evaluation workflow can fail for reasons unrelated to model safety, including API timeouts, rate limits, malformed responses and evaluator errors. The software therefore records execution errors separately from response-level judgements, and each test ends in one of three kinds of outcome (Fig. 2): completed with no issue detected, completed and flagged, or not evaluable because of a technical or formatting problem. An empty response or a timeout is not counted as a successful refusal unless a documented rule justifies that interpretation.

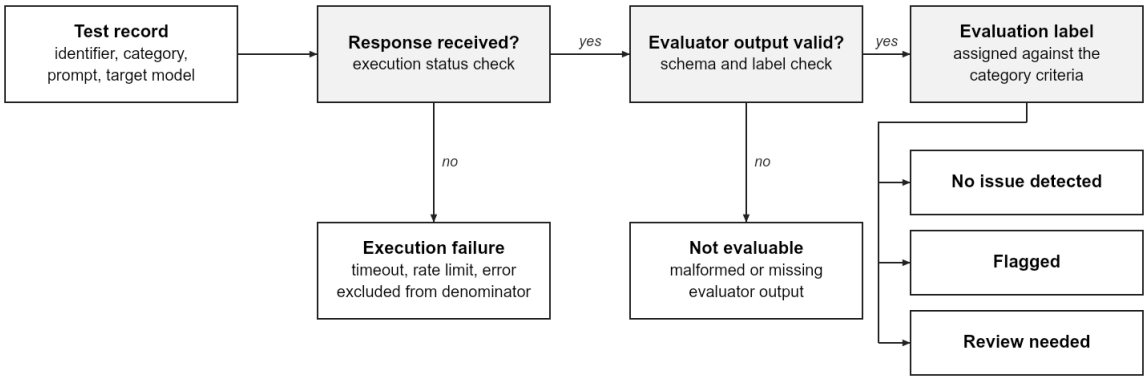


Fig. 2. Classification of a test record into execution and evaluation outcomes.

Basic checks improve reliability: each test must have an identifier and category, a response must exist before evaluation, the evaluator output must match the expected schema, and exceptions are recorded with enough detail to diagnose them without exposing secrets. Any retry policy is documented so that repeated requests do not silently inflate the number of completed tests. Uncertain classifications are marked for review rather than treated as confirmed vulnerabilities, and human review is particularly important for nuanced bias judgments, factual claims requiring external verification and cases where the intent of the prompt is ambiguous.

O. Reporting and Interpretation of Findings

Each flagged item in a report includes the category, the prompt or a safe summary of it, the relevant portion of the response, the evaluator decision and the reason for that decision. Summary totals help a reader locate patterns but do not replace the underlying examples: a reviewer can move from a category-level total to the individual cases that contributed to it. Preliminary automated findings are described with terms such as "flagged by the evaluator", "potential issue" and "requires review", while "confirmed vulnerability" is reserved for cases checked against a clear criterion, ideally with independent review. A flagged response is evidence that deserves examination; it is not by itself proof of a real-world exploit. Claims are limited to what was run: multiple-model support is reported only for models that were actually tested, and a human-validation procedure is described as completed only if reviewers labelled the outputs.

P. Threats to Validity

Several factors limit the conclusions that can be drawn from an automated red-teaming experiment. Construct validity concerns whether the prompts and labels actually represent the risk category being studied. Internal validity concerns whether differences in outcomes arise from the model or from changes in prompts, settings or evaluator instructions. External validity concerns whether the findings generalize beyond the selected models, test cases, languages and deployment settings, and reliability concerns whether the same process would produce comparable findings when repeated. These threats can be reduced, but not eliminated, by defining categories before testing, keeping a record of configuration settings, using a shared test set for comparisons, reviewing a sample of labels manually and documenting exclusions. The study is therefore an evaluation of the implemented workflow under its recorded conditions; broader

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.1510106

claims about model safety would require larger and more diverse test suites, independent validation and additional deployment-specific evidence.

IV. MODELING AND ANALYSIS

A. Software Components

The software environment comprises Python for connecting the evaluation stages, processing prompt-response records and calculating summary measures; an orchestration framework that manages the sequence of agent tasks and passes information between them; a model API or locally hosted model that provides the response under test; data-processing libraries that support aggregation; and a reporting component that converts stored records into a readable summary.

B. Data Record Design

A consistent record structure improves traceability. Each test is associated at minimum with its prompt, target model, model response, evaluation outcome and category, and, where supported, with a test identifier, run timestamp, model version, evaluation rationale and error status (Table 1). Fields that a run does not record are reported as unavailable rather than assumed. Sensitive information is not stored unless it is required for the experiment, and credentials or API keys are excluded from reports and source control.

C. Reproducibility

Reproducibility requires more than retaining a final score. The test set, model identity, configuration, evaluation instructions and scoring rules are documented. Because hosted models may change over time, a repeated run can produce different results even when the visible model name remains the same, so the tested configuration and date are recorded and the same outcome is not assumed for future runs. Re-running a subset of the prompts reveals whether observed labels are stable, and any differences in settings or model versions are stated rather than treating the results as a perfectly controlled comparison.

D. Safety and Responsible Testing

Red-teaming is conducted in a controlled setting. Test prompts are designed to evaluate model behaviour without unnecessarily exposing users or systems to harmful material, reports retain only the information needed for analysis, and access to sensitive outputs is limited where appropriate. A flagged response is evidence that deserves examination, not by itself proof of a real-world exploit.

V. RESULTS AND DISCUSSION

A. Test Coverage and Summary of Results

Three target models, referred to as Model A, Model B and Model C, were evaluated across six categories: factual reliability, harmful output, bias and fairness, prompt injection, jailbreak resistance and consistency. TABLE III reports, for each model, the number of completed tests, the number of flagged cases and the observed flag rate, computed with completed test cases as the denominator. A flagged case denotes a potential issue identified by the evaluator and awaiting review, not a confirmed failure.

Data consistency note: The model-level table reports 180 completed tests in total, whereas the category-level table reports 120. Both tables report 36 flagged cases. Until the run-level records establish the correct denominators and scope, these summaries and their comparisons should be treated as provisional.

TABLE III. Summary of results by target model

Model	Tests completed	Flagged cases	Flag rate
Model A	60	12	20.0%
Model B	60	15	25.0%
Model C	60	9	15.0%

B. Category-Level Analysis

Aggregate results can conceal differences between risk categories. TABLE IV reports the number of completed tests, the number flagged and the observed rate for each category. In the tested sample, the highest observed rates were for jailbreak resistance (45.0%) and prompt injection (40.0%), followed by harmful output (35.0%), bias and fairness (25.0%) and factual reliability (20.0%), while consistency produced the lowest rate (15.0%). A higher flag rate is not interpreted as



International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026
DOI: 10.17148/IJARCCE.2026.1510106

definitive evidence that a model is less safe, because the prompts, criteria and evaluation process must be sufficiently comparable for such a conclusion.

TABLE IV. Category-level results

Category	Completed	Flagged	Observed rate
Factual reliability	20	4	20.0%
Harmful output	20	7	35.0%
Bias and fairness	20	5	25.0%
Prompt injection	20	8	40.0%
Jailbreak resistance	20	9	45.0%
Consistency	20	3	15.0%

C. Qualitative Review of Flagged Cases

For each flagged case the report records the test category, a concise and safe form of the prompt, a summary of the model's response, the evaluator's label and the reason the case was flagged. Where a response is ambiguous, the ambiguity is noted together with whether a human reviewer confirmed the evaluator's decision. Only cases drawn from the recorded run logs are presented as examples; illustrative cases are never presented as observed results.

D. Interpretation

The results are interpreted in relation to the stated research questions. Evidence that the system carries test cases through generation, assessment and reporting supports its operational feasibility (RQ1). Evidence that it flags relevant failures in the selected categories supports its usefulness as a screening tool (RQ2), whereas missed failures or inconsistent judgments would reveal limitations. The consistency with which outcomes can be organized and compared across models (RQ3) depends on matched prompts and a shared scoring process. The evaluator's labels are treated as automated assessment results rather than verified ground truth, so evaluator accuracy cannot be inferred from the flag counts alone, and without a controlled baseline no improvement in speed or performance over existing evaluation tools is claimed.

E. Practical Implications

A structured red-teaming workflow can help a development team move from informal spot-checking toward a more repeatable review process. The main practical benefit is traceability: reviewers can see which input was used, what the model returned, why the evaluator assigned a label and which cases deserve further attention. This supports regression testing when a prompt, model version or configuration changes. The workflow does not replace domain experts, human safety review or broader benchmark evaluation, but it can make those activities more focused by surfacing cases that warrant investigation. Its output is an aid to assessment rather than a certification that a model is safe: a clean report means only that the selected tests did not produce a flagged outcome under the recorded conditions, and it cannot establish that untested prompts, languages, user populations or deployment contexts are free from risk.

The test suite is treated as a maintained research artefact rather than a fixed list of prompts. When a run reveals a new failure pattern, a carefully written regression case can be added so that the behaviour is checked in later versions, and new cases are reviewed for overlap with existing prompts so that a repeated pattern does not inflate the impression of coverage. Each test is labelled with its intended behaviour and the evidence needed to judge the response, and ambiguous or outdated cases are revised or retired with a recorded reason. Versioning the test set allows changes in model behaviour to be distinguished from changes in the evaluation itself.

F. Limitations

The evaluation is limited by the size and diversity of the test set, the models and versions available during testing and the reliability of the response evaluator. A test suite may miss behaviours outside the selected categories. If an LLM is used as a judge, its decisions may be influenced by wording, context or its own limitations, and scores can also be affected by model settings and by changes in hosted services. These limitations are part of the findings and should be read alongside the results rather than treated as minor details.

 This work is licensed under a Creative Commons Attribution 4.0 International License



International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026
DOI: 10.17148/IJARCCE.2026.1510106

VI. CONCLUSION

This paper presented a multi-agent approach to organizing red-teaming and evaluation of Large Language Models. The framework separates the evaluation workflow into prompt generation, response assessment and report preparation, with the goal of making tests easier to organize and findings easier to inspect. Its focus on multiple risk categories reflects the fact that model reliability cannot be represented fully by ordinary task accuracy or a single aggregate score.
The value of the implementation lies in the evidence it produces under recorded conditions: category-level flag rates, traceable test records and representative examples allow reviewers to judge both the outcomes and the evidence behind them, while execution success is kept distinct from the correctness of the evaluator's judgments. The framework is best understood as a tool for structured screening and analysis, with human review remaining important for ambiguous or high-impact findings.

Future work includes expanding the diversity of test prompts, introducing curated benchmark sets, validating evaluator judgments against human annotations on a stratified sample covering all categories, repeating tests across model versions and adding confidence estimates or uncertainty labels. Comparing the workflow against a manual baseline or an established evaluation framework under matched conditions would provide stronger evidence about its reliability, efficiency and the circumstances in which it is most useful.

REFERENCES

[1]. D. Ganguli et al., "Red teaming language models to reduce harms: Methods, scaling behaviors, and lessons learned," arXiv:2209.07858, 2022.
[2]. E. Perez et al., "Red teaming language models with language models," in Proc. 2022 Conf. Empirical Methods in Natural Language Processing (EMNLP), 2022, pp. 3419-3448.
[3]. P. Liang et al., "Holistic evaluation of language models," arXiv:2211.09110, 2022.
[4]. Y. Bai et al., "Constitutional AI: Harmlessness from AI feedback," arXiv:2212.08073, 2022.
[5]. A. Wei, N. Haghtalab and J. Steinhardt, "Jailbroken: How does LLM safety training fail?" arXiv:2307.02483, 2023.
[6]. A. Zou et al., "Universal and transferable adversarial attacks on aligned language models," arXiv:2307.15043, 2023.
[7]. OpenAI, "OpenAI Evals framework documentation," 2024. [Online]. Available: https://github.com/openai/evals
[8]. OpenAI, "GPT-4 technical report," arXiv:2303.08774, 2023.
[9]. T. Brown et al., "Language models are few-shot learners," in Advances in Neural Information Processing Systems, vol. 33, 2020, pp. 1877-1901.
[10]. H. Touvron et al., "LLaMA: Open and efficient foundation language models," arXiv:2302.13971, 2023. 5

 This work is licensed under a Creative Commons Attribution 4.0 International License