Cronos: Automating Bug Report Validation with LLMs using RAG Approach

Alay Nascimento
Sidia Institute of Science and Technology
Manaus, Brazil
alay.nascimento@sidia.com

Ana Paula Silva
Sidia Institute of Science and Technology
Manaus, Brazil
silva.ana@sidia.com

Lennon Chaves
Sidia Institute of Science and Technology
Manaus, Brazil
lennon.chaves@sidia.com

Abstract

In a testing team within a software institute, the validation of bug reports as valid or invalid is a critical but time-consuming activity. Performed manually, this validation is subject to errors, consumes effort and time, and competes with other activities of the testing team, especially when this team is under high demand, resulting in a backlog of bug reports to be validated. This study presents Cronos, an LLM-based tool with Retrieval Augmented Generation (RAG) that classifies bug reports as Valid or Not Valid, combining the retrieval of similar bug reports and relevant test cases. For this, we used 100 real bug reports collected and balanced between the Valid and Not Valid classes, submitted to the open source LLM models GPT, Qwen, and Gemma. Additionally, we used a also balanced knowledge base of 1526 bug reports and 133 test cases, retrieved through RAG with BERT embeddings and cosine similarity. We evaluated the models using the metrics Accuracy, Precision, Recall, and F1-Score, and statistical tests Cochran's Q test and McNemar post-hoc tests with Holm adjustment. The results indicate that the Qwen model achieved superior performance compared to the others, with Accuracy of 82%, F1-Score of 79.55% for Valid and 83.93% for Not Valid. GPT presented a Recall of only 46% for Valid, classifying 54% of the valid bug reports as invalid, while Gemma achieved Precision of 100% for Valid but Recall of only 58%. Cochran's Q test indicated a significant difference between the models with p-value = 0.0206, however the pairwise tests did not identify significant differences after the Holm adjustment. Therefore, the results of this study demonstrate that it is possible to use LLMs with RAG to automate the validation of bug reports in the industry, with Qwen being the most suitable model for balancing Precision and Recall for both classes.

Keywords

Bug Report Validation, Large Language Models, Retrieval Augmented Generation, Software Testing, Automated Classification

1 Introduction

During the Software Development Life Cycle (SDLC), bug reports are frequently opened so that developers can act on maintenance as soon as possible [18]. However, software maintenance consumes more than 70% of the total cost of the SDLC of a product, with defect correction and software evolution being the predominant activities in this process [18]. Large-scale projects use failure reporting and triage systems to manage defect reports, but an aggravating factor persists: many submitted reports are illegitimate duplicates or describe non-defects, studies indicate that up to 36% of the reports are duplicates or invalid [18].

In the context of a software testing team, failures are frequently registered and the evaluation of these failures consumes time and effort from developers [4, 5], being an essential part of modern software engineering. Bug reports contain structured information (operating system, version, attachments) and, mainly, unstructured content in natural language: Observed Behavior, Steps to Reproduce, and Expected Behavior [2, 3, 15]. Developers consider this information the most useful for triage and failure correction [29], however they are frequently absent or of low quality, analyses indicate that only 51.4% of bug reports describe Steps to Reproduce, and 35.2% describe Expected Behavior [2]. It is necessary for the testing team to evaluate whether the bug report is valid or not, so that developers do not waste time with invalid bug reports. Missing and incorrect information is the main cause of delay in failure correction [29].

To automate the process of Bug Report validation into valid or invalid failures registered by the testing team, we propose Cronos, a tool based on Large Language Models (LLMs) with Retrieval Augmented Generation (RAG) that classifies Bug Reports as valid or invalid. Cronos integrates a prompt engineering pipeline with retrieval of similar bug reports and relevant test cases, directing the model's reasoning through a structured decision process in 6 steps. Previous works addressed the prediction of bug report quality through statistical models based on superficial features [18], the CUEZILLA tool that measures quality and suggests improvements [29], and the identification of discourse patterns to detect missing information [2]. However, these approaches do not explore the potential of LLMs with RAG for the automatic classification of valid and invalid bug reports in the industrial context.

To evaluate Cronos, we used a dataset of 100 real bug reports registered by the testing team of a software institute, balanced between the Valid and Not Valid classes, validated by a professional with 7 years of experience. We evaluated three open source LLMs (GPT, Qwen, and Gemma) on the task of classifying each bug report as valid or invalid, comparing their results through classification metrics (Accuracy, Precision, Recall, and F1-Score) and statistical tests (Cochran's Q test and McNemar post-hoc tests with Holm adjustment).

The contributions of this work are: (1) Cronos, an LLM-based tool with RAG for automatic classification of bug reports as valid or invalid in an industrial context; (2) a comparative evaluation of three open source LLMs on this task, with performance analysis and statistical significance; and (3) evidence on the feasibility of using LLMs with RAG for automating the validation of bug reports by software testing teams.

2 Background

2.1 Bug Report Validation

Bug reports are the primary form of communication between testers and developers to ensure better coverage and maintenance of systems [8]. So that end users do not encounter problems when using a software product, any defects found during development must be reported and corrected before the release, which is why the software testing stage is essential [18]. A bug report contains structured characteristics that describe the failure and how to reproduce it, usually including steps to reproduce, stack trace, test cases, observed behavior, screenshots, expected behavior, code examples, summary, version, and error reports [8].

However, even though these are fundamental characteristics of bug report writing, not all bug reports contain the necessary information for developers to work on the correction, and, at times, the information provided may be incorrect [29]. A survey conducted with developers from the Apache, Eclipse, and Mozilla projects revealed that the main problem found in bug reports is the lack of completeness in the information provided [29]. In addition, several studies evidence errors in the description of reproduction steps, duplicate issues, use of incorrect software version, and lack of clarity in the expected and observed behaviors, which, according to developers, delay the failure correction process [29].

The validation of a bug report is a process that determines whether a bug report is *valid* or *invalid*, being a critical step in failure triage, in which developers or members of the testing team analyze the received reports and verify their entire scope and writing to determine which should be prioritized for correction [18]. In practice, a large number of bug reports are submitted daily in large-scale projects, for example, in Mozilla, on average 307 new bug reports are submitted per day [14]. Of these, from 22% to 79% have resolution DUPLICATE, INVALID, WORKSFORME, or INCOMPLETE, being classified as invalid [14]. The manual determination of the validity of a bug report is a time-consuming and tedious task, and invalid bug reports increase the difficulty of triage, wasting time and effort of developers [14].

There are some proposals that aim to predict the quality of bug reports such as Hooimeijer and Weimer [18] who made a model based on superficial features available at the time of submission, capable of functioning as an automatic filter and Zimmermann et al. [29] who developed CUEZILLA, which measures quality and suggests improvements, and Chaparro [2] identified discourse patterns to detect missing information. Zanetti et al. [25] proposed the use of collaboration networks to determine valid failures, extracting nine features based on the ASSIGN and CC relations of bug reports and using SVM as a classifier.

Despite these advances, none of these works explored the reasoning capability of the LLM in validating bug reports as invalid or valid, a gap that this work aims to fill.

2.2 LLM for Bug Report Validation

The growth of LLMs enables improvements in the process of reporting and validating failures. Choi and Yang [6] proposed AgentReport, a multi-agent pipeline based on LLMs for automated bug report generation. AgentReport integrates QLoRA-4bit lightweight fine-tuning, structured prompts based on the CTQRS framework, Chain-of-Thought reasoning, and one-shot exemplar, organized into seven modules. Evaluated with 3966 summary-report pairs from Bugzilla, AgentReport achieved 80.5% in CTQRS, 84.6% in ROUGE-1 Recall, 56.8% in ROUGE-1 F1, and 86.4% in SBERT, surpassing the baseline in all metrics [6].

Zhao et al. [28] proposed AnyPoC, a multi-agent framework for automated generation of proofs of concept (PoCs) that serves as a validation oracle for bug reports. AnyPoC produced 1.3× more valid PoCs for true reports and rejected 9.8× more false reports compared to state-of-the-art agents. To date, AnyPoC has discovered 122 new failures, with 105 confirmed and 86 already corrected [28].

3 Valid and Not Valid Bug Report

In a Software testing team, all opened bug reports are classified and closed with specific categories that determine their validity. The classification of a bug report as *Valid* or *Not Valid* depends on the closing category assigned, following the process defined by the team. Table 1 summarizes the closing categories and their validity criteria.

Table 1: Summary of closing categories and validity criteria

Category Type	Validity	Criterion
Defect Fixed	Valid	Real defect corrected
Known Limitation	Valid	Documented limitation of the system
Third Party Issue	Valid	External component problem
Test Error/Mistake	Not Valid	Error in test execution
Insufficient Defect	Not Valid	Insufficient information for reproduction
Irreproducible	Not Valid	Failure could not be reproduced
Concept/Requirement	Depends	Valid if the team had no prior knowledge of the requirement; Not Valid otherwise
Network Issue	Depends	Valid if retested with technical evidence; Not Valid otherwise
Duplicated	Depends	Not Valid if duplicate from the same team; Valid if from another team or requested by configuration variation

As observed in Table 1, the classification as *Valid* or *Not Valid* depends on contextual information. There are criteria to be followed to validate the effectiveness of a closed Bug Report, such as the prior knowledge of the testing team about the requirement, the existence of retesting with technical evidence, or the origin of the duplicated failure. This complexity makes manual classification a process that requires experience and knowledge of the domain, increasing the risk of inconsistencies among different analysts.

Figure 1 illustrates the manual process of bug report validation performed by the testing team. The process begins when the tester executes the tests on the device and registers the found failure, filling in the structured fields of the bug report. Then, the bug report is closed by the responsible developer in a specific category (e.g., Fixed, Duplicated, Network Issue). The bug report analyst then performs the manual validation, verifying the scope and writing of the report, as well as the team's prior knowledge about the involved requirements. Based on this analysis and the validity rules from Table 1, the analyst classifies the bug report as Valid or Not Valid and provides a justification. However, this manual process faces challenges, validation competes with other activities of the analyst,

such as support to the testing team and metrics analysis activities that can cause delays, and the daily volume of reports makes the process time-consuming and prone to errors. To mitigate these problems, we propose Cronos, an LLM-based tool with RAG that automates the classification of bug reports, reducing the analyst's manual workload and accelerating the validation process.

4 Methodology

This section presents the methodology of our study, organized into two parts. In Section 4.1, we describe the development methodology of Cronos, a RAG-based tool for bug report validation, covering the dataset composition, prompt engineering techniques, and LLM model selection. In Section 4.2, we detail the evaluation methodology of Cronos, including performance metrics and the experimental study design with the statistical tests used to verify significant differences between the models. Figure 2 presents the overview of the proposed methodology.

4.1 Methodology for Cronos Development

4.1.1 Dataset. To compose our dataset, we collected real bug reports from the software institute, already classified by the testing team. The final corpus gathered 100 samples collected with 50 classified as "Valid" and 50 classified as "Not Valid", all from tests on mobile devices in the scopes of sanity, regression, acceptance, and network services. Each bug report contains fields such as ID, title, problem description, steps to reproduce, expected result, medium resolution, small resolution, closing, development verification type, failure cause, applied countermeasure, tag, device model name, software version, classification (Valid/Not Valid), and classification justification. These data served as input for the model prompts.

4.1.2 Prompt Engineering. To insert the models into the context of bug report classification and guide their responses in a structured manner, we applied prompt engineering techniques. We organized the prompt design as follows:

- **System prompt:** definition of the model's role as classifier, presentation of 10 patterns of non-valid bug reports, decision process in 6 steps, and a JSON response template.
- **User prompt:** context of relevant test cases, bug report fields (except classification and justification), similar bug reports with their classification and justification performed by bug report analysts, and standardization of responses using a json schema.

To direct the response of the models, we inserted a decision process in the system prompt consisting of the following 6 steps:

- **Step 1:** analyze the bug report using the available fields (title, problem description, steps to reproduce, expected result, medium resolution, small resolution, closing, development verification type, failure cause, applied countermeasure, tag, device model name, and software version).
- **Step 2:** read the justification of each similar bug report.
- **Step 3:** identify the root cause pattern in the justification of each similar bug report.
- **Step 4:** compare the characteristics of the bug report with the patterns of the similar bug report.
- **Step 5:** if the bug report matches a "Not Valid" pattern from the similar bug reports, then classify as "Not Valid".
- **Step 6:** if no "Not Valid" pattern is found, then classify as "Valid"

The implementation of the user prompt incorporates the Retrieval Augmented Generation (RAG) technique [13], combined with BERT embeddings [10] and cosine similarity calculation [12] to retrieve similar bug reports, as represented in Figure 3. The system accesses a knowledge base with 1526 bug reports, retrieving the 8 most similar to the bug report to be classified. In parallel, it retrieves the 3 most relevant test cases among the 133 in the test case base.

We analyzed the distribution of bug reports by classification type in the knowledge base and identified an imbalance in the knowledge base. To balance this base, we employed synthetic data generation techniques focused on the "Not Valid" class, including synonym replacement, random swap, random deletion, phrase replacement, and shuffle sentence order [24, 27]. After balancing, the new composition of the knowledge base is detailed in table 2.

Table 2: Quantity of bug reports by classification before and after balancing the knowledge base

Class	Quantity Before	Quantity After
Valid	796	796
Not Valid	236	730

4.1.3 LLMs. The selection of LLM models for this study was guided by three criteria:

- The model must have an open source license, due to the institute's confidentiality policies.
- Compatibility with the available internal computational infrastructure.
- Computational impact related to the number of parameters [26].

We present the selected models in table 3, with their respective characteristics: name, version, parameters, size, and license.

Table 3: Configuration specifications of the LLM models selected in this study

Name	Version	Parameters	Size (GB)	Licence Type
GPT	gpt-oss	120B	65.3	Open Source
Qwen	qwen3.6	35B	23.8	Open Source
Gemma	gemma4	31B	17.4	Open Source

During the classification of bug reports, we adopted a temperature of 0.3 and top_p of 0.95 for the models evaluated in this study. The reduction of temperature decreases the entropy in the token probability distribution [17], conferring greater determinism to the outputs [21, 22], as desired for this binary classification task with a predefined response format.

4.2 Methodology for Cronos Evaluation

4.2.1 LLM Evaluation. We evaluated the performance of the models using the metrics Accuracy, Precision, Recall, and F1-Score.

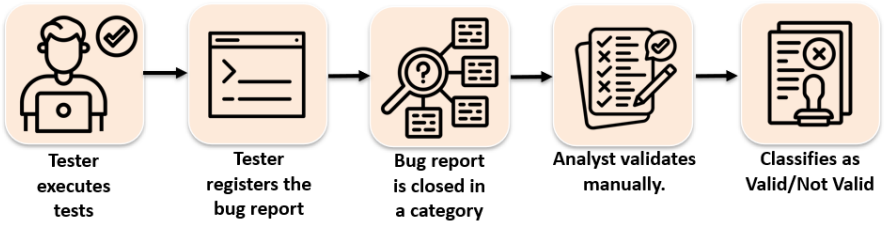

Figure 1: Manual Bug Report Validation

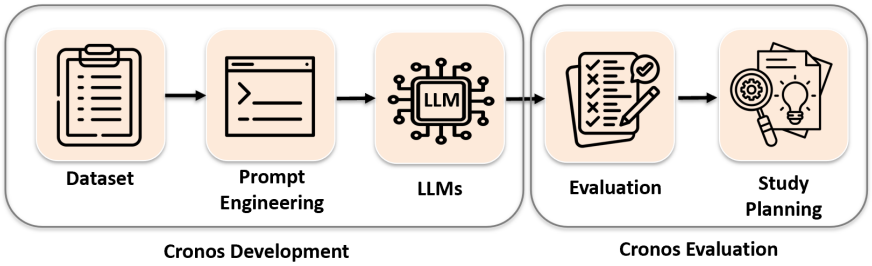

Figure 2: Methodology for developing and evaluating Cronos, a RAG-based tool for bug report validation

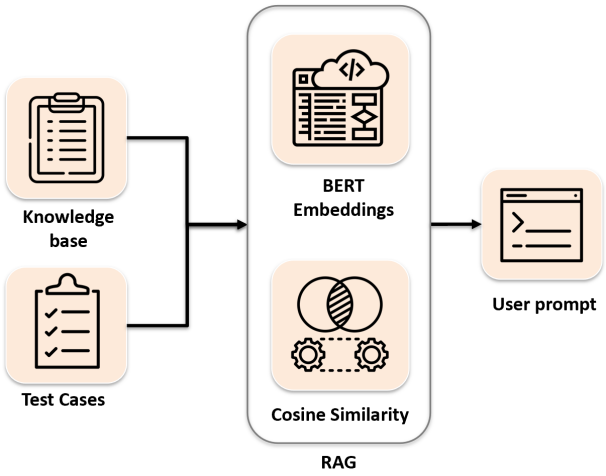

Figure 3: RAG Workflow for Bug Report Classification

Precision, presented in Equation (1), calculates the proportion of true positives in relation to all positive classifications. Accuracy, whose formula is detailed in Equation (2), measures the fraction of correct classifications in the total set of classifications [11].

$$\textbf{Precision} = \frac{TP}{TP + FP} \tag{1}$$

$$\textbf{Accuracy} = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{2}$$

In a complementary way, we employed the Recall and F1-Score metrics to measure, respectively, the detection capability of positive cases and the balance between Precision and Recall [20], as per Equations (3) and (4), respectively.

$$\textbf{Recall} = \frac{TP}{TP + FN} \tag{3}$$

$$\textbf{F1-score} = 2 \times \left(\frac{Precision \times Recall}{Precision + Recall} \right) \tag{4}$$

Where:

- TP: quantity of true positives.
- TN: quantity of true negatives.
- FP: quantity of false positives.
- FN: quantity of false negatives.

4.2.2 Study Planning. Here, we define the elements that compose our study, specifying the research scope and context, formulating the hypotheses, and selecting the variables and subjects:

1) Scope: We defined the objective of this study based on the GQM (Goal, Question, Metric) approach [1], as presented:

"**Analyze** the performance of 3 different LLM models for classifying bug reports **with the purpose of** evaluating **with respect to** proportion of correct classifications in the classification task **from the point of view of** researchers **in the context of** software testing team in a Software Institute".

2) Hypotheses:

- **H0**: There is no difference between the proportions of correct classifications of the GPT, Qwen, and Gemma models.
- **H1**: There is a difference between the proportions of correct classifications of the GPT, Qwen, and Gemma models.

We chose the proportion of correct classifications as the comparison metric between the models due to its capacity to measure classifier performance [23].

3) Context: To investigate the performance of the models in the automatic classification of bug reports, we designed this experiment, in which 100 real bug reports were classified by the GPT, Qwen, and Gemma models. We evaluated the results based on the

proportion of correct classifications and, from this proportion, we applied statistical tests to identify significant differences between the models.

4) Variable selection: We defined the variables as follows:

(1) Independent variables: The GPT, Qwen, and Gemma models.
(2) Dependent variable: The classification assigned by each model to each bug report, represented as a binary variable, where 1 indicates correct classification and 0 indicates incorrect classification.

5) Subject selection: We used as subjects of the experiment a dataset of 100 real bug reports, balanced and employed as an oracle, of which 50 were classified as "Valid" and 50 as "Not Valid". The validation of all classifications was conducted by a professional with 7 years of experience in the bug report classification activity within the testing team.

6) Selection of statistical tests: We defined the strategy of our study based on the nature of the data. As the data are binary (correct or error) and each bug report was classified by the three models, we selected Cochran's Q test [7] to verify the null hypothesis of equality of proportions of correct classifications among the three models. In case of rejection of the null hypothesis, we proceeded with McNemar post-hoc tests [19] to identify pairwise differences. The McNemar test compares the disagreements between two classifiers by means of a 2×2 contingency table. Considering that performing multiple pairwise comparisons increases the probability of Type I error [9], we applied the Holm adjustment [16], preventing the increase in the number of comparisons from elevating the probability of rejecting any true hypothesis [9, 16]. All tests were conducted with a significance level of $\alpha = 0.05$.

5 Experiments and Results

Based on the methodology detailed in Section 4, we evaluated the bug report classifications performed by the GPT, Qwen, and Gemma models. This section is organized into two subsections: in Section 5.1, we analyze the performance of the models through the metrics of Accuracy, Precision, Recall, and F1-Score. In Section 5.2, we apply Cochran's Q test and McNemar post-hoc tests with Holm adjustment to verify whether the observed differences are statistically significant, and we discuss the practical implications of the results for model selection.

5.1 Performance

We submitted the dataset to the prompt of each model, as detailed in Section 4, and evaluated their performances using Accuracy, Precision, Recall, and F1-Score. We present the results in table 4.

Table 4 presents the performance results of the GPT, Qwen, and Gemma models on the bug report classification task. We observed that the Qwen model achieved the highest accuracy of 82%, followed by Gemma with 79% and GPT with 71%.

Analyzing the "Valid" class, we highlight that Gemma achieved the highest precision of 100%, indicating that all classifications assigned as "Valid" were correct. However, in the Recall metric for this class the model obtained 58%, revealing that the model did not detect 42% of the valid bug reports. The Qwen model, in turn, presented the best balance between Precision (92.11%) and Recall (70%) for the "Valid" class, resulting in the highest F1-Score value for this class of 79.55%. The GPT model, although it obtained a Precision of 92%, achieved the lowest Recall for the "Valid" class at 46%, indicating that 54% of the valid bug reports were incorrectly classified as "Not Valid".

Regarding the "Not Valid" class, Gemma achieved a Recall of 100%, identifying all invalid bug reports, however with a Precision of 70.42%. The GPT model also obtained high Recall in this class, reaching 96%, although with the lowest precision of 64%. Again, Qwen obtained the best F1-Score for "Not Valid" with 83.93%, with Precision of 75.81% and Recall of 94%, demonstrating a balanced performance between the two classes.

Overall, Qwen presented the best global performance, achieving the highest F1-Score values for both classes. Gemma stood out for its Precision of 100% in the "Valid" class and Recall of 100% in the "Not Valid" class, however with lower recovery capability of valid bug reports. GPT obtained the lowest overall performance, with a negative highlight for the low Recall of the "Valid" class, demonstrating difficulty in identifying valid bug reports.

The metrics presented in table 4 indicate performance variations among the evaluated models. However, we cannot infer the superiority of one model over the others considering only these values. In order to obtain a more rigorous understanding of the results, we need to apply statistical tests that allow us to verify whether the observed differences are in fact significant.

5.2 Statistical Test

To verify whether the performance differences observed among the models are statistically significant, we adopted a significance level of $\alpha = 0.05$ and applied Cochran's Q test [7], used to compare three or more treatments with paired binary data (correct/incorrect) on the same test cases. We constructed a 100×3 matrix presented in table 5, where each row represents a bug report and each column represents a model (GPT, Qwen, and Gemma), with value 1 for correct classification and 0 for incorrect classification.

Cochran's Q test resulted in a Q statistic = 7.76 with 2 degrees of freedom, producing $p - value = 0.020651$. Since $p - value < \alpha$, we reject the null hypothesis that the models have the same proportion of correct classifications, indicating that there is a statistically significant difference between at least two of the three evaluated models.

To identify which pairs of models differ significantly, we conducted McNemar post-hoc tests [19] with Holm adjustment [16] for multiple comparisons. The $2x2$ contingency tables for each pairwise comparison are presented in table 6, indicating the concordant and discordant cases between the models. The results with raw $p - value$, without adjustment, denoted as $p - value_{raw}$, indicated a significant difference only between GPT and Qwen ($p - value_{raw} = 0.0192$), with 15 cases classified correctly by Qwen but incorrectly by GPT, while only 4 cases were classified correctly by GPT but incorrectly by Qwen. The comparisons between GPT and Gemma ($p-value_{raw} = 0.0768$) and between Qwen and Gemma ($p - value_{raw} = 0.6072$) were not significant. However, after applying the Holm adjustment, none of the pairwise comparisons reached statistical significance, as shown in table 7: GPT vs. Qwen ($p-value_{adj} = 0.0576$), GPT vs. Gemma ($p-value_{adj} = 0.1536$), and Qwen vs. Gemma ($p - value_{adj} = 0.6072$, where adjusted $p - value$,

Table 4: Models Performance Results

Model	Accuracy	Precision (Valid)	Recall (Valid)	F1-Score (Valid)	Precision (Not Valid)	Recall (Not Valid)	F1-Score (Not Valid)
GPT	71.00%	92.00%	46.00%	61.33%	64.00%	96.00%	76.80%
Qwen	82.00%	92.11%	70.00%	79.55%	75.81%	94.00%	83.93%
Gemma	79.00%	100.00%	58.00%	73.42%	70.42%	100.00%	82.64%

Table 5: Classification Results Matrix (100×3)

ID	GPT	Qwen	Gemma	ID	GPT	Qwen	Gemma
1	0	0	1	51	1	1	1
2	1	0	1	52	0	1	1
3	1	1	0	53	0	1	1
4	1	1	1	54	0	0	0
5	0	1	0	55	1	1	1
6	1	1	1	56	0	1	1
7	1	1	1	57	1	1	1
8	1	1	1	58	1	1	1
9	1	1	1	59	1	1	1
10	1	1	1	60	1	1	1
11	1	1	1	61	1	1	1
12	0	1	0	62	1	1	1
13	0	1	0	63	1	1	1
14	0	0	0	64	1	1	1
15	0	0	0	65	1	1	1
16	1	1	1	66	1	1	1
17	1	1	1	67	1	1	1
18	0	0	0	68	1	1	1
19	0	1	0	69	1	1	1
20	0	0	0	70	1	1	1
21	0	0	0	71	1	1	1
22	0	1	0	72	0	1	1
23	0	0	0	73	1	1	1
24	0	1	0	74	1	1	1
25	1	0	1	75	1	1	1
26	0	0	0	76	1	0	1
27	1	1	1	77	1	1	1
28	0	1	1	78	1	1	1
29	1	0	0	79	1	1	1
30	1	1	1	80	1	1	1
31	1	1	1	81	1	1	1
32	1	1	1	82	1	1	1
33	1	1	0	83	1	1	1
34	1	1	1	84	1	1	1
35	1	1	1	85	1	1	1
36	1	1	1	86	1	1	1
37	1	1	1	87	1	1	1
38	0	0	1	88	1	1	1
39	0	0	0	89	1	1	1
40	1	1	1	90	1	1	1
41	0	1	1	91	1	1	1
42	1	1	0	92	1	1	1
43	0	1	1	93	1	1	1
44	0	0	0	94	0	1	1
45	0	1	1	95	1	1	1
46	0	0	0	96	1	1	1
47	1	1	1	97	1	1	1
48	1	1	1	98	1	1	1
49	1	1	1	99	1	1	1
50	0	0	1	100	1	1	1

denoted $p - value_{adj}$, corresponds to the $p - value$ adjusted by the Holm method). This result demonstrates that, although Cochran's Q test indicates a global difference between the models, the pairwise tests cannot identify the difference with statistical significance after the adjustment for multiple comparisons.

Given this, the analysis of classification metrics becomes necessary to guide the choice of the viable model for the bug report classification activity. Thus, the Qwen model presented balanced performance between the two classes, with an F1-Score of 79.55% for Valid and 83.93% for Not Valid, and an overall Accuracy of 82%. In contrast, GPT obtained a Recall of only 46% for the Valid class, which means that 54% of Valid failures were classified as Not Valid. In the context of bug report classification, this balance is important because both types of error have distinct operational consequences: classifying a Valid failure as Not Valid (false negative) results in incorrect calculation of the testing team's effectiveness KPI, decreasing its value, in addition to preventing evidence of new concepts from being shared with the team. On the other hand, classifying a Not Valid failure as Valid (false positive) also alters the testing team's effectiveness KPI, and generates unnecessary effort by sharing documents and knowledge already existing in the team. Therefore, it is desirable that the model has both high Precision and Recall for both classes, minimizing both types of error. Qwen, by maintaining a Precision of 92.11% and Recall of 70% for Valid, and Precision of 75.81% and Recall of 94% for Not Valid, is the model that best balances the preservation of KPIs and the reduction of operational effort waste.

Table 6: Contingency Tables for Pairwise McNemar Tests with Holm Adjustment

GPT vs Qwen			
	Correct	**Incorrect**	**Total**
GPT Correct	67	4	71
GPT Incorrect	15	14	29
Total	82	18	100
$p - value_{raw} = 0.0192$		$p - value_{adj} = 0.0576$	
GPT vs Gemma			
	Correct	**Incorrect**	**Total**
GPT Correct	67	4	71
GPT Incorrect	12	17	29
Total	79	21	100
$p - value_{raw} = 0.077$		$p - value_{adj} = 0.154$	
Qwen vs Gemma			
	Correct	**Incorrect**	**Total**
Qwen Correct	73	9	82
Qwen Incorrect	6	12	18
Total	79	21	100
$p - value_{raw} = 0.607$		$p - value_{adj} = 0.607$	

6 Discussions

The positioning of this study in relation to the existing literature can be summarized by the evolution of approaches for bug report evaluation. Zanetti et al. [25] employed a method that uses collaboration network metrics with Support Vector Machine (SVM) to

Table 7: Pairwise McNemar Tests with Holm Adjustment

Comparison	$p - value_{raw}$	$p - value_{adj}$	Significant ($\alpha = 0.05$)
GPT vs Qwen	0.0192	0.0576	No
GPT vs Gemma	0.0768	0.1536	No
Qwen vs Gemma	0.6072	0.6072	No

classify bug reports as valid or invalid. More recently, Choi and Yang [6] proposed AgentReport for automated bug report generation from raw inputs, and Zhao et al. [28] developed AnyPoC for validation via proofs of concept. Our study complements these works by proposing Cronos, which combines LLMs with RAG and a structured decision process in 6 steps in the prompt to classify bug reports as Valid or Not Valid, filling the gap of approaches that explore the contextual reasoning of LLMs for validation in an industrial context.

The experimental results demonstrated that the Qwen model achieved the best performance compared to the GPT and Gemma models, with an Accuracy of 82% and balanced F1-Score for both classes. Cochran's Q test indicated a significant difference between the models ($p\text{-}value = 0.0206$), but the McNemar post-hoc tests with Holm adjustment did not identify significant pairwise differences, possibly due to the lower power of the individual tests after the adjustment for multiple comparisons [9].

An analysis of the behavior of the models reveals that GPT classified 54% of the valid bug reports as invalid, with a Recall of 46% for the Valid class. Gemma presented a Precision of 100% for Valid and Recall of 100% for Not Valid, however with a Recall of only 58% for Valid, demonstrating that the model is always correct when it classifies as Valid, but ends up not identifying 42% of the valid bug reports. Qwen maintained balance between the classes with Precision of 92.11% and Recall of 70% for Valid, and Precision of 75.81% and Recall of 94% for Not Valid. In the industrial context, this balance is fundamental because false negatives and false positives impact the effectiveness of the testing team in distinct ways. By minimizing both impacts, Qwen demonstrates to be the most suitable model for automation in Cronos.

The automation of bug report validation through LLMs with RAG provides:

- **Reduction of manual errors in triage**: By decreasing the dependence on manual evaluation, the occurrence of incorrect classifications is reduced.
- **Agility in bug report validation**: Acceleration of the process of identifying valid and invalid bug reports, allowing the testing team to prioritize valid bug reports.
- **Preservation of testing team effectiveness**: Balanced classification minimizes both false negatives and false positives, avoiding alterations in the effectiveness of the bug reports registered by the testing team.
- **Reduction of operational effort waste**: The decrease in false positives avoids unnecessary sharing of documents and knowledge already existing in the testing team.

7 Limitations of the Study

The results of this study are subject to the following limitations:

- The confidentiality policy of the software institute where we conducted the research prevents access to and disclosure of the data.
- The bug reports in the knowledge base and the test cases used are specific to the development context of the studied testing team, making it impossible to generalize the results to other teams in the institute.
- The size of the knowledge base, composed of 1526 bug reports and 133 test cases, does not allow the generalization of the obtained results.
- The absence of context about the internal institute repository files related to the mobile device models, which could provide additional relevant context for the classification of bug reports, and are used during their generation as reference documents for the expected behavior of the devices.

8 Conclusions

In this study, we presented Cronos, an LLM-based tool with RAG for automatic classification of bug reports as Valid or Not Valid in an industry environment. By submitting a set of 100 real bug reports to the prompt of each model, we verified that Qwen achieved superior performance compared to the others, with Accuracy of 82%, F1-Score of 79.55% for Valid and 83.93% for Not Valid. GPT presented a Recall of only 46% for the Valid class, classifying 54% of Valid failures as Not Valid. Gemma achieved Precision of 100% for Valid and Recall of 100% for Not Valid, however with a Recall of 58% for Valid. We validated the results through Cochran's Q test, which indicated a significant difference between the models with $p\text{-}value = 0.0206$. However, the McNemar post-hoc tests with Holm adjustment did not identify significant differences between the pairs of models. Given this, the analysis of classification metrics indicated Qwen as the most balanced model, minimizing both the alteration in the testing team's effectiveness and the waste of operational effort.

Regarding industrial application, Cronos contributes significantly to the reduction of manual errors in classification and to agility in bug report validation. In future research, we intend to include new bug reports, with the aim of expanding the knowledge base during the retrieval of similar bug reports through RAG. Furthermore, we seek to integrate the generation of the classification justification into the model and incorporate explainability techniques into the process, clarifying the reasons why a bug report was assigned to a given class. Finally, we plan to conduct a controlled experiment with the testing team, to evaluate the strengths and limitations of the model implementation in the testing team, and investigate its applicability in real scenarios.

Artifact Availability

Owing to the confidentiality of the project data and the stipulations of the Non-Disclosure Agreement (NDA), the artifacts and data sets used in this study cannot be made publicly available.

Acknowledgements

This paper is a result of the Research, Development & Innovation Project (ASTRO) performed at Sidia Institute of Science and Technology sponsored by Samsung Eletrônica da Amazônia Ltda., using

resources under terms of Federal Law No. 8.387/1991, by having its disclosure and publicity in accordance with art. 39 of Decree No. 10.521/2020.

References

[1] Victor R Basili. 1994. Goal, question, metric paradigm. *Encyclopedia of software engineering* 1 (1994), 528–532.

[2] Oscar Chaparro. 2017. Improving Bug Reporting, Duplicate Detection, and Localization. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. 421–424. doi:10.1109/ICSE-C.2017.27

[3] Lennon Chaves, Davi Gonzaga, Leonardo Tiago, Ana Silva, and Flávia Oliveira. 2025. Automatic Generation of Bug Reports Using Large Language Models: An Evaluation in a Software Institute. In *Anais do XXIV Simpósio Brasileiro de Qualidade de Software* (São José dos Campos/SP). SBC, Porto Alegre, RS, Brasil, 337–345. doi:10.5753/sbqs.2025.13599

[4] Lennon Chaves, Flávia Oliveira, and Leonardo Tiago. 2024. Automating issue reporting in software testing: Lessons learned from using the template generator tool. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*. 278–282.

[5] Lennon Chaves, Flavia Oliveira, and Leonardo Tiago. 2024. Enhancing Automated Tools: Reporting Bugs with Bug Builder. In *Anais do IX Simpósio Brasileiro de Testes de Software Sistemático e Automatizado* (Curitiba/PR). SBC, Porto Alegre, RS, Brasil, 80–82. doi:10.5753/sast.2024.3674

[6] Seojin Choi and Geunseok Yang. 2025. AgentReport: A Multi-Agent LLM Approach for Automated and Reproducible Bug Report Generation. *Applied Sciences* 15, 22 (2025). doi:10.3390/app152211931

[7] William G Cochran. 1950. The comparison of percentages in matched samples. *Biometrika* 37, 3/4 (1950), 256–266.

[8] Steven Davies and Marc Roper. 2014. What's in a bug report?. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. 1–10.

[9] Janez Demšar. 2006. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research* 7, Jan (2006), 1–30.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*. 4171–4186.

[11] Edna Dias Canedo and Bruno Cordeiro Mendes. 2020. Software requirements classification using machine learning algorithms. *Entropy* 22, 9 (2020), 1057.

[12] Abdullah Dogan, Alev Mutlu, and Pinar Karagoz. 2016. Cosine Similarity-Based Pruning for Concept Discovery. In *International Symposium on Computer and Information Sciences*. Springer, 90–96.

[13] Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. 2024. A survey on rag meeting llms: Towards retrieval-augmented large language models. In *Proceedings of the 30th ACM SIGKDD conference on knowledge discovery and data mining*. 6491–6501.

[14] Yuanrui Fan, Xin Xia, David Lo, and Ahmed E. Hassan. 2020. Chaff from the Wheat: Characterizing and Determining Valid Bug Reports. *IEEE Transactions on Software Engineering* 46, 5 (2020), 495–525. doi:10.1109/TSE.2018.2864217

[15] Davi Gonzaga, Leonardo Tiago, Ana Silva, Flávia Oliveira, and Lennon Chaves. 2025. Improving Bug Reporting by Fine-Tuning the T5 Model: An Evaluation in a Software Industry. In *Anais do X Simpósio Brasileiro de Testes de Software Sistemático e Automatizado* (Recife/PE). SBC, Porto Alegre, RS, Brasil, 141–143. doi:10.5753/sast.2025.13591

[16] Sture Holm. 1979. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics* (1979), 65–70.

[17] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. [n. d.]. The Curious Case of Neural Text Degeneration. In *International Conference on Learning Representations*.

[18] Pieter Hooimeijer and Westley Weimer. 2007. Modeling bug report quality. In *Proceedings of the 22nd IEEE/ACM international conference on Automated software engineering*. 34–43.

[19] Peter A Lachenbruch. 2005. McNemar Test. *Encyclopedia of Biostatistics* 5 (2005).

[20] Alay Nascimento, Flávia Oliveira, Leonardo Tiago, and Lennon Chaves. 2025. Who Should Test the Requirement? A Comparative Study on Requirements Classification for Assigning Test Teams using the Pre-Trained Models. In *Simpósio Brasileiro de Engenharia de Software (SBES)*. SBC, 671–677.

[21] Mario Patrício, Silas Eufrásio, Anderson Uchôa, Lincoln S Rocha, Daniel Coutinho, Juliana Alves Pereira, Matheus Paixão, and Alessandro Garcia. 2025. 404: Civility Not Found? Evaluating the Effectiveness of Small Language Models in Detecting Incivility in GitHub Conversations. In *Simpósio Brasileiro de Engenharia de Software (SBES)*. SBC, 304–314.

[22] Rylan Schaeffer, Joshua Kazdan, and Yegor Denisov-Blanch. 2025. Min-p, Max Exaggeration: A Critical Analysis of Min-p Sampling in Language Models. *arXiv preprint arXiv:2506.13681* (2025).

[23] Marina Sokolova and Guy Lapalme. 2009. A systematic analysis of performance measures for classification tasks. *Information processing & management* 45, 4 (2009), 427–437.

[24] Jason Wei and Kai Zou. [n. d.]. EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks. ([n. d.]).

[25] Marcelo Serrano Zanetti, Ingo Scholtes, Claudio Juan Tessone, and Frank Schweitzer. 2013. Categorizing bugs with social networks: A case study on four open source software communities. In *2013 35th International Conference on Software Engineering (ICSE)*. 1032–1041. doi:10.1109/ICSE.2013.6606653

[26] Biao Zhang, Zhongtao Liu, Colin Cherry, and Orhan Firat. [n. d.]. When Scaling Meets LLM Finetuning: The Effect of Data, Model and Finetuning Method. In *The Twelfth International Conference on Learning Representations*.

[27] Xiang Zhang, Junbo Zhao, and Yann LeCun. 2016. Character-level Convolutional Networks for Text Classification. arXiv:1509.01626 [cs.LG] https://arxiv.org/abs/1509.01626

[28] Zijie Zhao, Chenyuan Yang, Weidong Wang, Yihan Yang, Ziqi Zhang, and Lingming Zhang. 2026. AnyPoC: Universal Proof-of-Concept Test Generation for Scalable LLM-Based Bug Detection. *arXiv preprint arXiv:2604.11950* (2026).

[29] Thomas Zimmermann, Rahul Premraj, Nicolas Bettenburg, Sascha Just, Adrian Schroter, and Cathrin Weiss. 2010. What Makes a Good Bug Report? *IEEE Transactions on Software Engineering* 36, 5 (2010), 618–643. doi:10.1109/TSE.2010.63