BODHIVRUKSHA JOURNAL OF DIVERSE DISCIPLINE
E-ISSN : 3139-1486
An Open Access, Peer-Reviewed, Multidisciplinary Scholarly Journal
Bi-monthly | Multilingual (English, Hindi, Marathi) | Academic Research
@ Volume 2, Issue 5 (September-October 2026) (Special Issue 1) ISSN: 3139-1486 (Online)

# Small Language Models for Automated Software Bug Report Classification and Summarization

**Amruta Sangtani**

Department of Artificial Intelligence and Data Science
Ajeenkya DY Patil School of Engineering
Pune, Maharashtra, India Academic Year: 2026–2027 Email: amruta.sangtani@dypic.in

**Prof. Pooja Dehankar**

Department of Artificial Intelligence and Data Science
Ajeenkya DY Patil School of Engineering
Pune, Maharashtra, India Email: poojadehankar@dypic.in

**Dr. Vishal Puranik**

Technical Seminar Coordinator Email: vishalpura@gmail.com

## Abstract

Before a bug report can be assigned to anyone, somebody has to decide what kind of problem it describes, how serious it is, how soon it needs attention and which component it touches. In busy projects this triage work piles up, and two people reading the same report often disagree. Large Language Models (LLMs) classify text well, but they are costly to run and awkward to deploy for everyday triage. Small Language Models (SLMs) need far less hardware, yet nobody has shown that they cope with the noisy, code-filled text of real bug reports. This paper reviews ten recent studies and proposes a design in which an SLM suggests labels and a short summary, a rule-based layer checks them, and reports the model is unsure about go to a human triager. We also set out how the design should be evaluated. Nothing has been implemented, so the paper makes no claim about accuracy or speed.

**Keywords:** Small Language Models, Bug Report Classification, Software Maintenance, Bug Triage, Bug Report Summarization, Zero-Shot Classification, Knowledge Distillation, Large Language Models.

## Introduction

Wherever software has users, bug reports follow. A typical report carries a title, a description, steps to reproduce, perhaps a log or stack trace, and some metadata. Before work can begin, a triager reads it and answers a few questions: is this a real defect or a feature request, how severe is it, how urgent, and which component does it affect? This sorting is called bug report classification and is one part of bug triage. As the number of reports grows, it becomes a bottleneck, and the label a report receives can depend on who happened to read it. Earlier automation relied on classical machine learning and deep learning, for example

BODHIVRUKSHA JOURNAL OF DIVERSE DISCIPLINE
E-ISSN : 3139-1486
An Open Access, Peer-Reviewed, Multidisciplinary Scholarly Journal
Bi-monthly | Multilingual (English, Hindi, Marathi) | Academic Research
@ Volume 2, Issue 5 (September-October 2026) (Special Issue 1) ISSN: 3139-1486 (Online)

CNN, LSTM and kNN models trained on projects such as Eclipse and Mozilla [8]. More recently LLMs have been tried on bug reports [1], [9], but they are expensive and usually reached through large infrastructure. Small Language Models look like a possible middle path, because studies outside the bug report domain say they can match or come close to larger models on classification while using fewer resources [2], [3]. Bug reports are short, informal, technical and often mixed with code, so whether that result carries over is still open.

**Small Language Models.** Language models built on the transformer architecture learn from large text corpora. Here, a small model means one with roughly a hundred million to a few billion parameters; one empirical study of zero-shot classification takes 77M to 3B parameters as its definition of small [2]. Such models need less memory and computation, so they can run on consumer-grade hardware [6] and, with suitable techniques, on devices [10].

**Problem and Motivation.** Classifying a bug report means making several linked decisions (type, severity, priority and component) from text that is noisy and unevenly written. We do not want to remove the triager. The goal is to hand them a quick first label from a model small enough to run locally or on-premises, and to check that label against explicit rules before anyone sees it.

This paper makes four contributions. First, it reviews recent work on SLMs, LLMs and bug report analysis.Second, it proposes a structured way to classify and summarize bug reports with an SLM. Third, it adds rule-based validation and human review around the model output. Fourth, it defines evaluation criteria and discusses limitations and future work. The paper is organized as follows. The Literature Review gives the theoretical background and Related Work surveys ten studies. The Problem Statement and Research Objectives follow, then the Proposed System Architecture and the Methodology. The Evaluation Framework, Expected Outcomes and Discussion, and Limitations and Future Work come next, and the Conclusion closes the paper.

Literature Review

**Bug reports and classification.** A bug report is a natural-language record of unexpected software behaviour. The usual classification targets are the report type (bug, feature request, question), the severity (how much the failure harms the user), the priority (how soon it should be fixed) and the component (the part of the system involved). Surveys find that categorization is the most common task in learning-based bug report analysis, followed by bug localization and severity prediction [8]. Related variants include configuration bug classification [9] and structured bug report generation [1]. **Adaptation of SLMs.** Four routes are common. In zero-shot prompting the model sees the report and a description of the labels, with no training examples [2], [6]. In few-shot prompting a handful of labelled examples are added to the prompt. In fine-tuning the weights are updated on labelled reports from the target domain, which can raise accuracy but costs data and effort [3]. In knowledge distillation a smaller student model is trained to imitate a larger teacher [5], although some reasoning ability can be lost along the way.

**Classification formulation.** Let $x$ be the processed text of a bug report and $\{c_1, \ldots, c_K\}$ a set of labels. Themodel gives a score $z_k$ for every label, and the probability of label $c_k$ is

BODHIVRUKSHA JOURNAL OF DIVERSE DISCIPLINE
E-ISSN : 3139-1486
An Open Access, Peer-Reviewed, Multidisciplinary Scholarly Journal
Bi-monthly | Multilingual (English, Hindi, Marathi) | Academic Research
@ Volume 2, Issue 5 (September-October 2026) (Special Issue 1) ISSN: 3139-1486 (Online)

$$P(c_k \mid x) = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$$

For generative SLMs the scores can come from the likelihood of the label tokens given a prompt, which is how scoring functions are compared in zero-shot studies [2].

**Deterministic validation.** Validation applies explicit rules that always give the same result for the same input. For classification, the rules can check that the output is one of the allowed labels, that labels for different fields agree with each other, and that the required fields were present in the report. Such checks cannot tell whether a severity level is right for the business.

Related Work

We reviewed ten papers on bug report classification and generation, small language model evaluation, multilingual and efficient models, and surveys of language models in software engineering. Table 1 gives the methodology, findings and limitations of each.

Language Models for Bug Report Analysis

Acharya studies bug reports with graph neural networks and instruction fine-tuned LLMs, covering severity prediction and the generation of structured reports [1]. The study shows that language models can be useful on bug reports, but it depends on large models and a high computational cost. Lopez-Duran et al. combine LLMs and encoders for configuration bug classification [9]. Long et al. review machine learning and deep learning approaches to bug report analysis and identify trends, datasets and challenges [8]; they also point out that heavier models such as BERT were underused because of their complexity.

Small Language Models for Zero-Shot Classification

Lepagnol et al. compare small and large models for zero-shot classification across many datasets and find the small ones on par with, or better than, larger ones in several cases [2]. The datasets are general text classification datasets, not bug reports, which is why the result needs to be checked on bug report text.

Comparing Large and Small Language Models Yu et al. compare LLMs and SLMs and report lower inference cost with competitive accuracy, at the price of domain-specific fine-tuning [3]. This is the central trade-off for triage: a cheaper model is attractive only if the effort of adapting it stays manageable.

Multilingual Prompting

Banerjee et al. use multilingual prompting with SLMs for query classification [4]. Projects with a global user base often receive reports in more than one language, so this matters, although the work was not evaluated on bug reports.

Knowledge Distillation

Wang et al. distil knowledge from LLMs into SLMs for summarization and note that some reasoning ability is lost [5]. That loss is a concern for severity and priority decisions, which depend on reasoning about impact.

SLMs in Software Engineering and On-Device Efficiency

BODHIVRUKSHA JOURNAL OF DIVERSE DISCIPLINE
E-ISSN : 3139-1486
An Open Access, Peer-Reviewed, Multidisciplinary Scholarly Journal
Bi-monthly | Multilingual (English, Hindi, Marathi) | Academic Research
@ Volume 2, Issue 5 (September-October 2026) (Special Issue 1) ISSN: 3139-1486 (Online)

Gheyi et al. apply SLMs to detect refactoring bugs at lower computational cost, but validation across projects is limited [6]. Zhang et al. survey language models for software engineering, including code understanding, bug detection and maintenance [7], and Xia et al. discuss efficient on-device language models [10].

Literature Survey

**Table 1.** Selected Literature Review

| Author / Year | Methodology Used | Key Findings | Limitations |
|---|---|---|---|
| Acharya (2025) [1] | Graph Neural Networks (GraphSAGE, GAT) and instruction fine-tuned LLMs | Improved bug severity prediction and generated structured bug reports. | High computational cost; focuses on LLMs rather than lightweight SLMs. |
| Lepagnol et al. (2024) [2] | Zero-shot classification using Small Language Models | SLMs achieved performance comparable to larger models with fewer resources. | Evaluated on general text classification datasets. |
| Yu et al. (2025) [3] | Comparative evaluation of LLMs and SLMs | SLMs reduced inference cost while maintaining competitive accuracy. | Requires domain-specific fine-tuning. |
| Banerjee et al. (2025) [4] | Multilingual prompting with SLMs | Improved multilingual query classification accuracy. | Not evaluated on software bug reports. |
| Wang et al. (2025) [5] | Knowledge distillation from LLMs to SLMs | Enhanced summarization using lightweight models. | Some reasoning capability lost. |
| Gheyi et al. (2025) [6] | SLMs for software refactoring bug detection | Efficient detection with lower computational cost. | Limited validation across projects. |
| Zhang et al. (2025) [7] | Survey of language models for software engineering | Reviewed code understanding, bug detection and maintenance. | Survey only; no implementation. |
| Long et al. (2025) [8] | Systematic review of ML/DL approaches | dentified trends, datasets and challenges. | Limited focus on lightweight SLMs. |
| Lopez-Duran et al. (2025) [9] | LLMs and encoders for configuration bug classification | Improved configuration bug classification. | Configuration-specific scope. |
| Xia et al. (2025) | Efficient on-device language model | Balanced efficiency and | Not focused on bug |

BODHIVRUKSHA JOURNAL OF DIVERSE DISCIPLINE
E-ISSN : 3139-1486
An Open Access, Peer-Reviewed, Multidisciplinary Scholarly Journal
Bi-monthly | Multilingual (English, Hindi, Marathi) | Academic Research
@ Volume 2, Issue 5 (September-October 2026) (Special Issue 1) ISSN: 3139-1486 (Online)

| | | | |
|---|---|---|---|
| [10] | techniques | deployment. | reports. |

Comparison of Research Directions

Table 2 sets the research directions side by side and notes what each means for the framework. It does not rank them by effectiveness.

**Table 2.** Qualitative Comparison of Research Directions

| Direction | Focus | Consideration for Framework |
|---|---|---|
| LLM-based bug analysis [1], [9] | Severity prediction, structured reports, configuration bugs | Shows value of language models; cost is high. |
| SLM zero-shot classification [2], [6] | Small models used with prompts only | Low-cost baseline; check on bug text. |
| LLM vs. SLM comparison [3] | Accuracy against inference cost | Report accuracy and cost together. |
| Multilingual prompting [4] | Query classification in several languages | Handle non-English reports. |
| Knowledge distillation [5] | Transfer from large to small models | Possible route to a stronger SLM. |
| On-device efficiency [10] | Efficient models for deployment | Supports local, private triage. |
| rveys and reviews [7], [8] | Trends, datasets, challenges | Source of datasets and metrics. |

**Synthesis.** Small models are shown to be efficient on general classification [2], [3], and language models are shown to help with bug reports [1], [9], but the two lines of work have rarely met. The surveys [7], [8] describe bug report analysis broadly, and the SLM studies [2], [4], [5], [10] are not centred on bug reports. The framework proposed here therefore studies lightweight SLMs on bug report classification directly and wraps their output in validation and human review. It is not meant to replace issue trackers or existing triage tools. What sets it apart is that the SLM label is treated as a suggestion, checked first by rules and then by a person. Whether this improves classification quality while lowering cost still has to be verified.

Research Gaps

**Limited evidence for SLMs on bug reports:** existing SLM studies cover general text classification

[2] or other software tasks [6], not bug report classification.

**Domain adaptation cost:** fine-tuning is often needed to keep accuracy [3], and labelled bug data may not exist for every project.

**Reasoning loss:** distilled or very small models may lose some reasoning ability [5], which can affect severity and priority decisions.

**Multilingual and noisy text:** multilingual prompting [4] has not been tested on bug reports that contain

BODHIVRUKSHA JOURNAL OF DIVERSE DISCIPLINE
E-ISSN : 3139-1486
An Open Access, Peer-Reviewed, Multidisciplinary Scholarly Journal
Bi-monthly | Multilingual (English, Hindi, Marathi) | Academic Research
@ Volume 2, Issue 5 (September-October 2026) (Special Issue 1) ISSN: 3139-1486 (Online)

code, logs and mixed languages.

**Narrow scope of existing studies:** work on configuration bugs [9] or refactoring bugs [6] does not show whether results generalize across projects.

**Evaluation practice:** surveys note missing statistical tests and few lightweight-model studies [8], and the reviewed papers offer no common benchmark that compares accuracy and cost on bug reports.

Problem Statement and Research Objectives

Classifying a bug report means making several linked decisions, namely type, severity, priority and component, from text that is noisy and unevenly written. In busy projects this triage work piles up, and the label a report receives can depend on who happened to read it. LLMs are costly to run and awkward to deploy for everyday triage, while SLMs have not been shown to cope with the code-filled text of real bug reports. The proposed system, "Small Language Models for Automated Software Bug Report Classification and Summarization", addresses this gap through the following objectives:

**Literature Review-** Review the literature on SLMs for text classification and for bug analysis.

**Framework Formulation-** Formulate an SLM-based classification and summarization framework for bug reports.

**Validation and Human Review-** Integrate rule-based validation and human review into the framework.

**Evaluation Criteria-** Formulate criteria for judging the accuracy, efficiency and reliability of SLM-based classification.

**Limitations-** State the limitations of the approach and the work needed before deployment.

These objectives are explored through four research questions. **RQ1:** How can small language models classify bug reports by type, severity, priority and component, and summarize them for triagers? **RQ2:** Which adaptation methods (zero-shot prompting, fine-tuning, knowledge distillation) suit SLMs best on bug report text? **RQ3:** Which metrics should decide the accuracy, efficiency and reliability of SLM-based classification? **RQ4:** What limitations must be overcome before the approach is deployed?

**Scope.** The work covers the classification and summarization stages of bug triage. It makes no claim about accuracy on a deployed system; it offers the concept, a literature review and an evaluation framework.

Proposed System Architecture

**Architecture Overview**

The proposed system keeps input, SLM inference, validation, output, review and storage separate. This makes every stage easy to test and shows how each part of the report ends up in the final label. Fig. 1 shows the six layers and the feedback loop from review back to the input and storage layers.

BODHIVRUKSHA JOURNAL OF DIVERSE DISCIPLINE
E-ISSN : 3139-1486
An Open Access, Peer-Reviewed, Multidisciplinary Scholarly Journal
Bi-monthly | Multilingual (English, Hindi, Marathi) | Academic Research
@ Volume 2, Issue 5 (September-October 2026) (Special Issue 1) ISSN: 3139-1486 (Online)

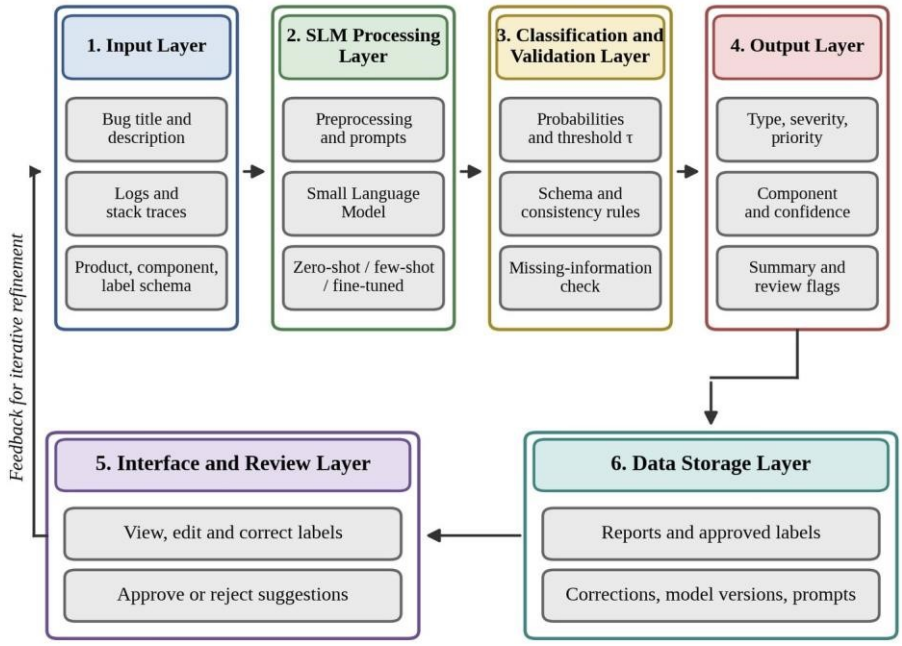



**Fig. 1.** Proposed SLM-based bug report classification and summarization framework.

Input Layer

The input layer collects reports and metadata from the tracker in a structured form, together with the label schema.

SLM Processing Layer

This layer runs preprocessing, builds the prompt and calls the SLM in zero-shot, few-shot or fine-tuned mode. Its output is stored apart from human-assigned labels.

Classification and Validation Layer

This layer applies the rule checks and the confidence threshold. A failure produces a warning that says what went wrong, for example naming the field whose label is not in the schema.

Output, Interface and Review Layers

The output layer presents the suggested labels, their probabilities, the short summary and the validation status, with low-confidence reports marked for review. The interface and review layer lets the triager view, correct and approve labels, and keeps a history so that suggested and approved labels can be compared.

Data Storage Layer and Workflow

The data storage layer holds reports, approved labels, corrections, model versions and prompts under access control. Corrected labels are the main source of data for later fine-tuning. As Fig. 1 shows, a report passes from input through SLM processing, validation and output to the review layer; corrections flow back to the input and storage layers.

BODHIVRUKSHA JOURNAL OF DIVERSE DISCIPLINE
E-ISSN : 3139-1486
An Open Access, Peer-Reviewed, Multidisciplinary Scholarly Journal
Bi-monthly | Multilingual (English, Hindi, Marathi) | Academic Research
@ Volume 2, Issue 5 (September-October 2026) (Special Issue 1) ISSN: 3139-1486 (Online)

Methodology

The method turns a raw bug report into a suggested classification in four steps: preprocessing, SLM inference, validation and review. Fig. 2 shows the eight stages in detail, and the subsections below describe each of them.

Input Collection

Reports are pulled from a tracker such as Bugzilla, GitHub or Jira. From each one we take the title, description, steps to reproduce, logs, product, component and any existing labels. Eclipse and Mozilla are the projects used most often in the literature [8], so they are natural sources of labelled data. A missing field is recorded as missing and never filled with a default value.

Preprocessing

The text is cleaned and normalized. Stack traces and code snippets are kept apart from the prose because they carry different information, and very long logs are truncated with a note. Duplicate and near-duplicate reports are flagged, and the language of each report is detected so that non-English reports can be handled on purpose [4].

Label Schema

The schema lists the allowed values for each target: report type, severity, priority and component. Because the schema is explicit, the validation layer can reject any output that is not a valid label.

BODHIVRUKSHA JOURNAL OF DIVERSE DISCIPLINE
E-ISSN : 3139-1486
An Open Access, Peer-Reviewed, Multidisciplinary Scholarly Journal
Bi-monthly | Multilingual (English, Hindi, Marathi) | Academic Research
@ Volume 2, Issue 5 (September-October 2026) (Special Issue 1) ISSN: 3139-1486 (Online)

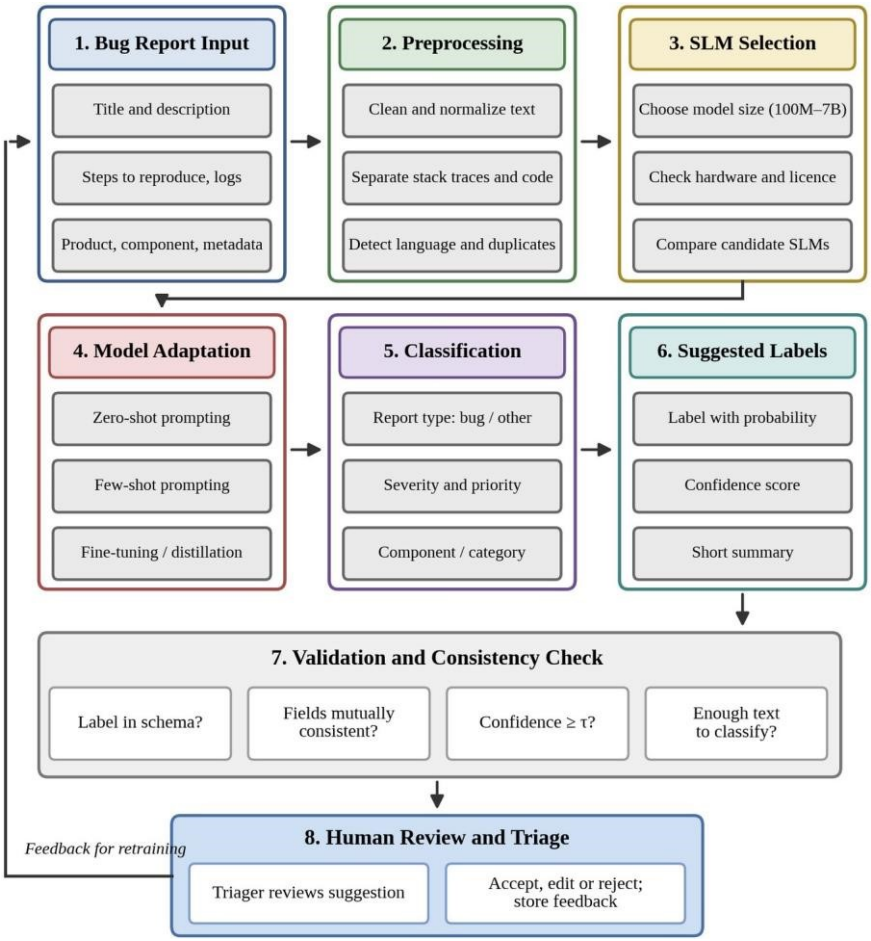



**Fig. 2.** Architecture of the proposed SLM-based bug report classification and summarization system.

SLM Selection and Adaptation

One or more SLMs are chosen according to size, licence and hardware limits. Three adaptation modes are compared: zero-shot prompting [2], few-shot prompting, and fine-tuning on labelled reports [3].

Distillation from a larger teacher stays as an optional route [5]. Prompts, model versions and settings are all recorded so that experiments can be repeated.

Classification and Confidence

For each target the SLM returns a label and a probability computed as in Eq. (1). A report is accepted automatically only when its highest probability reaches a threshold τ:

$$\text{route}(x) = \begin{cases} \text{accept}, & \max_k P(c_k \mid x) \geq \tau \\ \text{human review}, & \text{otherwise} \end{cases}$$

BODHIVRUKSHA JOURNAL OF DIVERSE DISCIPLINE
E-ISSN : 3139-1486
An Open Access, Peer-Reviewed, Multidisciplinary Scholarly Journal
Bi-monthly | Multilingual (English, Hindi, Marathi) | Academic Research
@ Volume 2, Issue 5 (September-October 2026) (Special Issue 1) ISSN: 3139-1486 (Online)

The value of τ is chosen on a validation set and trades automation against error.

Summarization

The same SLM is asked for a short summary covering the symptom, the conditions under which it occurs and the key log lines, so that a triager can grasp the case quickly. Summarization with lightweight models has been explored through distillation [5], and the quality and structure of bug reports have been studied with language models [1]. The prompt tells the model to use only information that is in the report.

Validation and Review

Before a label or summary is shown, the validation layer checks four things: the label is in the schema; type, severity and priority agree with each other (a feature request should not carry a crash severity); the report had enough text to classify; and the summary does not bring in component names or error terms that the report lacks. The triager then accepts, edits or rejects the suggestion, and any correction is stored for later retraining. Review stays in the loop because rules cannot judge how much a defect matters to the business.

Triage Workflow Algorithm

Algorithm 1 summarizes the workflow, and Fig. 3 gives an illustrative example of one report passing through it.

***Algorithm 1: SLM-Based Bug Report Triage Workflow***

***Input:*** *Bug report R, label schema S, SLM M, prompt template T, confidence threshold τ.*

1. *(fields ← CollectReport(R)); missing fields are marked as missing, never defaulted*
2. *(text ← CleanAndNormalize(fields))*
3. *(prose, traces ← SeparateTracesAndCode(text)); very long logs are truncated with a note*
4. *(dup, lang ← FlagDuplicates(text), DetectLanguage(text))*
5. *(prompt ← BuildPrompt(T, S, prose, traces))*
6. *for each target t in {type, severity, priority, component}: (z ← Score(M, prompt, S[t])); (p ← Softmax(z)); (label[t] ← argmax p)*
7. *(summary ← Summarize(M, prose, traces)), using only information that is in the report*
8. *(checks ← LabelInSchema ∧ FieldsConsistent ∧ EnoughText ∧ SummaryFaithful)*
9. *(route ← accept) if (checks) holds and the highest probability reaches τ; otherwise (route ← human review)*
10. *(PresentSuggestion(labels, probabilities, summary, checks, route))*
11. *(decision ← TriagerAcceptEditOrReject(suggestion))*
12. *(StoreCorrection(decision, model version, prompt, τ)) for later retraining*

BODHIVRUKSHA JOURNAL OF DIVERSE DISCIPLINE
E-ISSN : 3139-1486
An Open Access, Peer-Reviewed, Multidisciplinary Scholarly Journal
Bi-monthly | Multilingual (English, Hindi, Marathi) | Academic Research
@ Volume 2, Issue 5 (September-October 2026) (Special Issue 1) ISSN: 3139-1486 (Online)

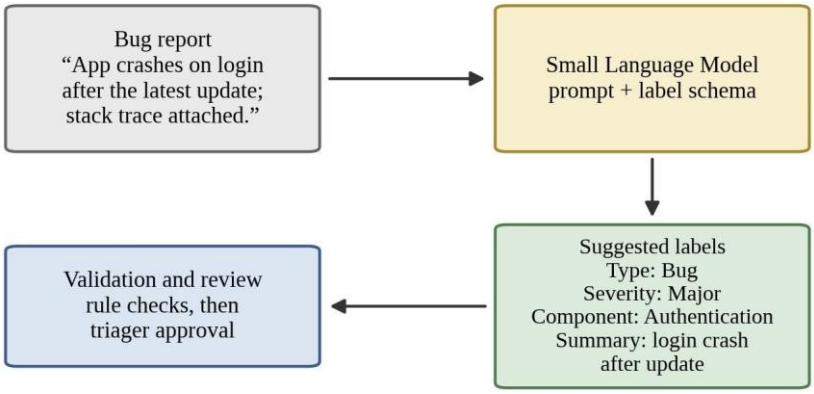



**Fig. 3.** Illustrative example of classifying a bug report (labels are for illustration only).

Proposed Evaluation Framework

The framework needs empirical evaluation, and the criteria below are offered for future work. For a single class, let TP, FP and FN be the numbers of true positives, false positives and false negatives.

Classification Metrics

Accuracy is the fraction of reports labelled correctly out of N reports in total:

$$\text{Accuracy} = \frac{N_{\text{correct}}}{N} \times 100$$

Precision and recall for a class are

$$P = \frac{TP}{TP + FP}, \qquad R = \frac{TP}{TP + FN}$$

The F1-score combines them:

$$F1 = \frac{2PR}{P + R}$$

Because bug classes are often imbalanced, macro-averaged F1 over K classes is reported too:

$$\text{Macro-F1} = \frac{1}{K} \sum_{k=1}^{K} F1_k$$

Efficiency, Automation and Summary Quality

Efficiency is measured by the mean inference latency over N reports,

$$\bar{t} = \frac{1}{N} \sum_{i=1}^{N} t_i$$

BODHIVRUKSHA JOURNAL OF DIVERSE DISCIPLINE
E-ISSN : 3139-1486
An Open Access, Peer-Reviewed, Multidisciplinary Scholarly Journal
Bi-monthly | Multilingual (English, Hindi, Marathi) | Academic Research
@ Volume 2, Issue 5 (September-October 2026) (Special Issue 1) ISSN: 3139-1486 (Online)

together with memory use, hardware used and cost per thousand reports. The automation rate is the share of reports accepted without review under Eq. (2), reported with the error rate among accepted reports. The validation pass rate is the proportion of outputs that satisfy the schema and consistency rules. Summary quality is judged by reviewers for faithfulness to the report and completeness of key details, and can be supported by overlap metrics such as ROUGE against reference summaries. Accuracy figures should always be read next to the cost figures, since a cheap model is only useful if its errors are acceptable.

**Experimental Design**

Future work could compare (1) a classical ML or deep learning baseline, (2) one or more SLMs in zero-shot and few-shot modes, (3) fine-tuned SLMs, and (4) a larger LLM as a reference, all on the same labelled reports from projects such as Eclipse and Mozilla, with the same train/test split and metrics. Cross-project testing should be included, and a statistical test should accompany the results, since many studies lack one [8]. Model versions, prompts, hardware and thresholds must be recorded. The paper does not claim any improvement in performance, because the framework has not been tested.

**Expected Outcomes and Discussion**

The framework is expected to give triagers a quick first label and a short summary for each report, with every suggestion checked by rules and by a person. Whether it improves classification quality while lowering cost has to be shown by experiments. The following current trends make the question worth testing.

**Lightweight and on-device models.** Work on efficient and on-device language models [10] makes it realistic to classify reports inside an organization, without sending them, and any private data they hold, to an outside service.

**Small versus large models.** Evidence that small models are competitive on classification [2], [3] is a reason to test them on bug reports, where the cost of classifying every incoming report adds up.

**Multilingual projects.** Open-source and global teams get reports in several languages, and multilingual prompting [4] is one way to cope.

**Distillation.** Passing knowledge from large to small models [5] may improve SLM quality, but the loss of some reasoning ability has to be measured.

**Human–AI collaboration.** The model suggests and the triager decides, so accountability stays with people.

Limitations, Ethical Considerations and Future Work

**Human Oversight:** The SLM label is a suggestion, never a final decision. The triager accepts, edits or rejects it, because rules cannot judge how much a defect matters to the business.

**Data Privacy:** Running a small model locally or on-premises avoids sending bug reports, which may contain private data, to an outside service. Reports, corrections, model versions and prompts are held under access control.

**Limitations:** The output can only be as good as the report: vague or very short reports give uncertain labels. SLMs may make mistakes that validation cannot fully catch, and labels in public trackers can themselves be inconsistent. The literature deals with language models in software engineering broadly rather than with SLMs on bug reports directly, and the framework itself is untested, so no gain in accuracy, speed or cost

BODHIVRUKSHA JOURNAL OF DIVERSE DISCIPLINE
E-ISSN : 3139-1486
An Open Access, Peer-Reviewed, Multidisciplinary Scholarly Journal
Bi-monthly | Multilingual (English, Hindi, Marathi) | Academic Research
@ Volume 2, Issue 5 (September-October 2026) (Special Issue 1) ISSN: 3139-1486 (Online)

has been established.

**Future Work:** The next step is a prototype that joins an SLM with a label schema, a validation engine and a review interface, evaluated on public bug datasets. Further directions include fine-tuning on corrections stored from triagers, distillation from a larger teacher [5], multilingual evaluation [4], extending the framework to severity and priority prediction, duplicate detection and structured report generation [1], and testing on real projects with reproducible prompts and honest reporting of failures.

## Conclusion

Small Language Models are a promising lightweight option for classifying bug reports, but a model that is efficient on general text is not automatically reliable on bug reports. This paper suggests a framework in which an SLM proposes labels, a validation layer checks them and a human triager approves them. Ten publications were reviewed, research directions were compared and an evaluation approach was outlined. The central idea is to pair the efficiency of small models with explicit checks, traceability and human judgement. Whether the approach is useful in practice has to be shown by experiments.

## References

1. J. Acharya and G. Ginde, “Can we enhance bug report quality using LLMs? An empirical study of LLM-based bug report generation,” arXiv preprint arXiv:2504.18804, 2025.
2. P. Lepagnol, T. Gerald, S. Ghannay, C. Servan, and S. Rosset, “Small language models are good too: An empirical study of zero-shot classification,” in Proc. LREC-COLING 2024, Torino, Italy, 2024, pp. 14923–14936.
3. H. Yu et al., “Comparative evaluation of large and small language models,” 2025.
4. P. Banerjee et al., “Multilingual prompting with small language models for query classification,” 2025.
5. L. Wang et al., “Knowledge distillation from large to small language models for summarization,” 2025.
6. R. Gheyi, M. Ribeiro, and J. Oliveira, “Evaluating the effectiveness of small language models in detecting refactoring bugs,” arXiv preprint arXiv:2502.18454, 2025.
7. Z. Zhang et al., “Unifying the perspectives of NLP and software engineering: A survey on language models for code,” arXiv preprint arXiv:2311.07989, 2025.
8. Long, J. Gong, H. Fang, and T. Chen, “Learning software bug reports: A systematic literature review,” arXiv preprint arXiv:2507.04422, 2025.
9. N. Lopez-Duran, D. R. Organvídez, F. Cruz, and D. Benavides, “Configuration bugs classification using LLMs and encoders,” in Proc. 29th ACM Int. Systems and Software Product Line Conf. (SPLC), vol. A, 2025, pp. 190–200, doi: 10.1145/3744915.3748477.
10. Xia et al., “Efficient on-device language model techniques,” 2025.