Annales Mathematicae et Informaticae
63 (2026) pp. 132–148
DOI: 10.33039/ami.2026.06.004
URL: https://ami.uni-eszterhazy.hu

Data cleaning and spam filtering methods in the TAWOS database

Márk Szabó[a], **Ádám Kovács**[ab], **Gábor Kusper**[a]

[a]Eszterházy Károly Catholic University
szabo.mark@uni-eszterhazy.hu, kovacs2.adam@uni-eszterhazy.hu,
kusper.gabor@uni-eszterhazy.hu

[b]University of Debrecen, Doctoral School of Informatics

Abstract. The TAWOS dataset is a large relational database of issue-tracking data mined from open-source projects. While it is valuable for empirical software engineering, the issue texts also contain spam, placeholders, and off-topic noise that can distort downstream analytics and fine-tuning tasks. We present a four-stage filtering pipeline for cleaning TAWOS and focus on a validity scoring step that assigns each issue a 0–100 score from its title and description.

We compare a deterministic rule-based classifier, OwnMetrics, against four local small language models (Llama-3.1-8B, Mistral-7B, Phi-3.5-mini, and Gemma-3-4B) prompted for JSON scores. Evaluation first uses a near-balanced labeled benchmark of 947 GitHub issues (481 spam/noise and 466 legitimate issues), collected from moderator-locked spam issues and legitimate issues from popular repositories.

On this GitHub-based threshold-selection benchmark, OwnMetrics obtains the highest accuracy, 91.0%, at threshold 75, while the best LLM configurations reach 89.2% (Gemma-3-4B) and 89.1% (Mistral-7B). A separate manually labeled in-domain validation on 300 Jira/TAWOS issues (39 spam/noise and 261 non-spam) provides an in-domain sanity check consistent with the benchmark results, with OwnMetrics giving the strongest spam-class F1 among the selected configurations.

On a 10,000-item sample, OwnMetrics yields zero parsing failures and an average execution time of 2.28 ms per item, whereas the LLMs require seconds per item and Meta-Llama-3.1-8B exhibits failure rates above 54%. Applying the full filtering pipeline to the processed TAWOS issue corpus flags 16.2% of records for filtering. The results show that a lightweight domain-tailored scorer can be both accurate and robust for large-scale issue cleaning.

Submitted: March 31, 2026
Accepted: June 30, 2026
Published online: July 17, 2026

Keywords: software repositories, issue tracking, data cleaning, spam filtering, large language models

2020 *Mathematics Subject Classification:* Primary 68N30; Secondary 68P15, 68T50

1. Introduction

Issue tracking systems are central to modern software development, because they collect bug reports, feature requests, status updates, discussions, and historical traces of the development process. Large public corpora built from issue trackers therefore provide a useful basis for empirical software engineering research. The TAWOS dataset is one such resource: it contains 508,963 issues from 44 open-source Agile projects and stores them in a relational schema designed for broad reuse [21].

However, large issue datasets are only as reliable as the textual and structural quality of the underlying reports. Prior work has shown that the quality of bug reports strongly affects downstream development activity and that issue misclassification can distort later modeling steps [12, 19, 22, 25]. More generally, issue-report analysis has long emphasized that software repositories contain heterogeneous, noisy, and inconsistently labeled artifacts [2, 11, 24]. For this reason, cleaning and validation are not only preprocessing conveniences: they are methodological requirements for trustworthy repository mining.

Our work addresses this need for the TAWOS database. The immediate motivation is practical: the cleaned data should provide a better basis for later teaching material, statistical analysis, simulation of development processes, and future fine-tuning workflows. We therefore design a filtering pipeline that scores issue validity on a 0–100 scale and removes records that fall below a selected threshold. The pipeline combines a hand-crafted rule-based method tailored to issue text with local LLM-based zero-shot scoring.

The main contributions of this paper are as follows. First, we formulate a practical issue-validity scoring task for TAWOS and construct a curated labeled benchmark of 947 issues. Second, we present OwnMetrics, a deterministic rule-based scoring method based on six interpretable components and an explicit early spam-detection stage. Third, we benchmark OwnMetrics against four local small language models across several temperature settings. Fourth, we analyze not only predictive quality but also output robustness, score distributions, cross-model agreement, spam detection rate, and execution time. Finally, we report the effect of applying the complete filtering pipeline to the processed TAWOS issue corpus.

2. Related work

2.1. Bug-report quality and issue-report structure

Bug and issue reports have long been studied as textual artifacts whose quality affects maintenance work. Zimmermann et al. analyzed what makes a bug report

useful for developers and emphasized the role of concrete observations, reproduction information, and contextual detail [25]. Soltani et al. later studied the significance of individual bug-report elements and showed that report usefulness depends not only on the presence of a report in an issue tracker, but also on the information carried by its textual fields [19]. These findings motivate validity signals such as title clarity, description detail, coherence, and technical vocabulary in our scorer.

A complementary line of work studies how issue trackers can improve report structure at submission time. Sülün et al. analyzed issue-template usage in large-scale GitHub projects and showed that structured templates and forms are widely used in popular repositories and are associated with improved issue-management outcomes [20]. Nikeghbal et al. proposed GIRT-Model for automatically generating issue-report templates, aiming to help maintainers define structured issue forms [17]. Our problem is different: we clean an already existing, historical issue corpus. Therefore, we cannot assume that the original reports followed a template; the method must evaluate noisy title–description pairs after the fact.

2.2. Issue report classification and label quality

Automated issue report classification is closely related to our work, but its usual target variable differs. Classical studies classify change requests or issue reports as bugs, enhancements, questions, documentation tasks, or other issue types [2, 11, 12]. Herzig et al. showed that bug/non-bug misclassification can distort downstream defect-prediction results [12], while Herbold et al. investigated the feasibility of predicting bug and non-bug issues automatically [11]. Recent work has extended this line with pre-trained and transformer-based models, and the systematic mapping study of Laiq and Dobslaw summarizes traditional machine learning, deep learning, transformer, and emerging LLM-based approaches for issue report classification [16].

This literature is useful for our task because it shows that issue title and description are standard inputs for text-based classification. At the same time, our output is not an issue type such as bug or enhancement. We estimate whether an issue should be retained for downstream corpus use. This makes explainability, scalability, and failure-free execution more central than in many interactive triage settings. Label quality is also important. Colavito et al. studied the impact of data quality for automatic issue classification using pre-trained language models and identified construct-validity problems in issue labeling [4]. Wu et al. similarly showed, in the context of security bug report prediction, that label correctness can substantially affect conclusions drawn from mined datasets [23]. These observations support treating our benchmark as a controlled spam/noise benchmark rather than as a complete ground truth for all levels of TAWOS issue quality.

2.3. Noise, spam, and validity threats in repository mining

Repository-mining studies have repeatedly warned that public repository data can be misleading without explicit validity checks. Kalliamvakou et al. discuss the

promises and perils of mining GitHub and emphasize that repository datasets require careful filtering before they are used for empirical claims [15]. Tu and Zhang also treat issue-tracking data quality as a measurable concern rather than as a guaranteed property of the dataset [22]. This perspective matches our motivation: cleaning is not only a preprocessing convenience, but a requirement for trustworthy repository mining.

Spam and off-topic issue reports are less studied than bug-versus-feature classification, but recent work on locked GitHub issues is relevant. Ferreira et al. analyzed GitHub issues locked as too heated and showed that moderation labels and actual discussion content do not always align perfectly [7]. Ehsani et al. released an annotated dataset of locked GitHub issue threads, demonstrating that locked issue data can support analyses of problematic repository interactions [6]. These works justify using moderation signals as useful proxies, while also showing why they should not be interpreted as perfect ground truth.

2.4. LLM-based and rule-based filtering methods

Large language models are increasingly used in software-engineering tasks. Hou et al. provide a systematic literature review of LLMs for software engineering and highlight both the promise of these models and the methodological challenges involved in evaluating them [13]. In issue-report classification, Rejithkumar et al. formulated the task as text-to-text generation [18], and Colavito et al. benchmarked LLMs for automated issue labeling, including trade-offs between performance, response time, hardware requirements, and response quality [3]. These studies motivate comparing against language-model baselines, but they also show why accuracy alone is insufficient for deployment-oriented filtering.

Rule-based methods remain useful when the target signal is dominated by stable and interpretable textual patterns. In the present task, obvious spam, placeholder reports, malformed descriptions, link flooding, missing software context, and very low-effort titles are often recognizable without open-ended reasoning. OwnMetrics therefore uses deterministic scoring rather than supervised training or unconstrained generation. Compared with supervised issue-type classifiers, it does not require a project-specific training set. Compared with LLM prompting, it has no parsing uncertainty and can be run cheaply over a large exported issue corpus.

2.5. Positioning of the present work

The present work is positioned at the intersection of bug-report quality assessment, issue report classification, repository-noise analysis, and LLM-based text filtering. Its specific goal is narrower than general bug triage: it estimates whether issue texts are suitable for downstream use in a cleaned TAWOS corpus. The main contribution is therefore not a new general-purpose classifier, but a deterministic, transparent, locally executable validity scorer evaluated against local zero-shot language-model baselines under accuracy, robustness, and runtime constraints.

3. TAWOS and the evaluation data

3.1. The TAWOS database

TAWOS (Tawosi Agile Web-based Open-Source) is a large issue-tracking dataset introduced by Tawosi et al. [21]. It was mined from 44 public Jira-based projects and stored as a relational database. The original dataset includes not only issue texts but also related entities such as comments, users, change logs, sprints, versions, components, and links between issues [21]. In the present work, however, we deliberately focus on the central `issues` table, because it contains the title, description, status, type, and time-related fields that are most directly relevant for first-pass validity scoring. Figure 1 shows the broader relational context of this table in the TAWOS database.

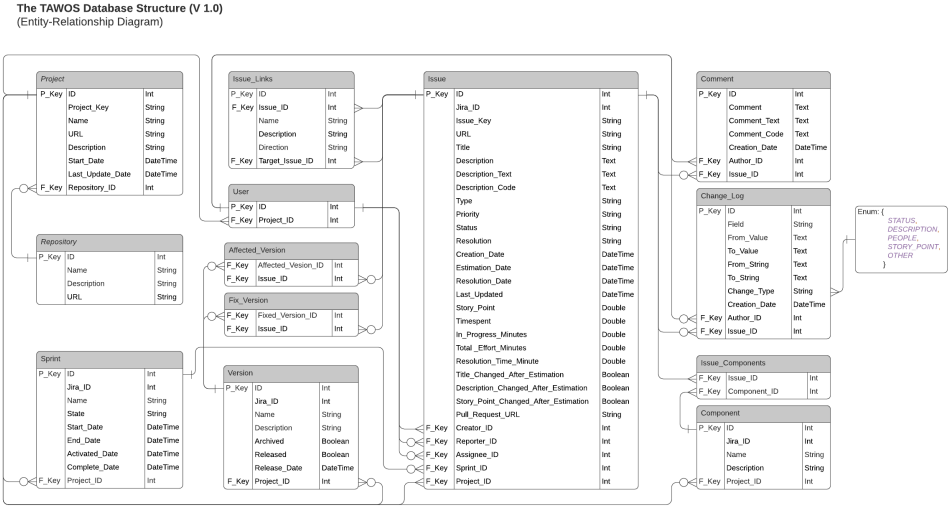

Figure 1. Structure of the TAWOS database.

This design choice keeps the present study focused and computationally lightweight. At the same time, it preserves a clear path for future work, where other data could be integrated into a richer spam/noise detector.

3.2. Cleaning pipeline

The overall workflow consists of four stages, summarized in Figure 2. First, the relevant issue fields are extracted from the original MySQL database and stored as CSV snapshots for the experiments. We used CSV because it is easy to inspect manually, convenient for debugging individual records, and simple to reuse across repeated scoring runs. Some field selection and structural filtering operations could also be expressed directly as SQL queries before export; future implementations

may push more of this structural filtering into SQL. We did not benchmark SQL-level processing against file-based processing, and no performance claim is made about this implementation choice. Second, a structural cleaning step removes irrelevant tables or columns for the present task. Third, each remaining issue receives a validity score in the range 0–100. Fourth, issues that fall below a selected threshold are filtered out together with their related data. The core of this paper is the third stage, i.e., how validity is defined and how reliably it can be estimated.

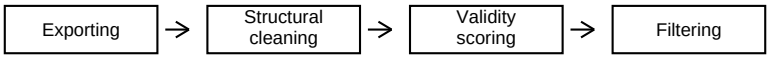

Figure 2. TAWOS issue-cleaning pipeline used in the present study.

3.3. Labeled benchmark

To evaluate the scoring methods, we assembled a curated labeled benchmark of 947 issues. The benchmark was near-balanced between 481 spam/noise and 466 non-spam examples. Spam examples were collected from GitHub issues in the 2023–2025 period by using the GH Archive event stream [9] and selecting issues from active, highly starred repositories that had been locked by moderators for spam-related reasons. Legitimate non-spam issues were retrieved with the GitHub REST API from popular and actively moderated repositories [10].

Although the target corpus is Jira-based and the benchmark is GitHub-based, both sources contain issue titles and descriptions written in the same general genre of software-development communication. This makes the benchmark suitable for a first controlled spam/noise comparison under identical labeled conditions. However, it should not be interpreted as a complete in-domain gold standard for TAWOS or as an unbiased estimate of Jira-specific generalization performance; this platform-shift threat is revisited in Section 7.

3.4. In-domain Jira/TAWOS validation set

To directly check this platform shift, we additionally created a small manually labeled in-domain validation set from Jira-based TAWOS issues. The set contains 300 issues, including 39 spam/noise cases and 261 non-spam cases. The issues were selected partly by random sampling from the processed TAWOS issue corpus and partly by manual search for suspicious records, so that the validation set would contain enough spam/noise cases for meaningful diagnostic metrics. Consequently, this set should not be used to estimate the natural prevalence of spam/noise in TAWOS. It was not used for prompt design, model selection, threshold selection, or tuning. Instead, it is used only as an in-domain sanity check for the configurations selected before this validation. Because the set is imbalanced, with 13.0% spam/noise cases, we report balanced accuracy and spam-class precision, recall, and F1 in addition to overall accuracy.

4. Validity scoring methods

4.1. Common scoring interpretation

All methods assign a validity score between 0 and 100, where lower values indicate spam, non-software-related, or very poor-quality reports, and higher values indicate clear and informative issue descriptions. The threshold-based binary decision is obtained afterward; the score itself is intended to remain useful even beyond binary filtering, for example when ranking issues for manual inspection.

4.2. OwnMetrics: a rule-based validity scorer

OwnMetrics is a deterministic rule-based classifier that uses only the issue title and description. Before scoring, the text is lightly normalized: URLs are removed, markdown links are reduced to their anchor text, markdown headers and HTML comments are stripped, and common template markers are ignored. This preprocessing is intended to reveal the actual human-authored content rather than its surrounding formatting.

The method then performs an early spam check. Extremely short placeholders, one-character content, template-only reports, and clearly promotional or abusive patterns are assigned a fixed low score immediately, without running the full scoring pipeline. In practice this catches obvious spam such as promotional messages, gambling or dating patterns, pharmacy-style advertisements, simple placeholders, and other trivial cases.

If the issue passes this early filter, OwnMetrics computes six partial scores. Table 1 summarizes their ranges and semantics.

The title score rewards detailed and structured titles, while penalizing very short, vague, or meaningless titles. The description score rewards longer and better formatted descriptions, and penalizes empty, joke-like, or template-only content. The coherence score checks whether the description actually elaborates on the same topic as the title. The style and professionalism scores down-rank visually spam-like, aggressive, or socially inappropriate reports. Finally, the technical-content score rewards domain-specific vocabulary, version strings, stack traces, code fragments, and other signals of genuine software discussion.

The raw score is the sum of the six partial scores and lies in the interval $[-130, 140]$. It is then normalized to the final 0–100 scale:

$$\text{score} = \text{clip}_{[0,100]}\left(\left\lfloor 100 \cdot \frac{r + 130}{270} \right\rfloor\right),$$

where r denotes the raw score and $\text{clip}_{[0,100]}$ truncates the result to the valid score range.

The main strength of OwnMetrics is that every component is interpretable and directly grounded in observable textual properties. This makes the method attractive when large-scale processing, debugging, or predictable behavior are more important than open-ended reasoning.

Table 1. OwnMetrics score components and their ranges.

Component	Range	Main signals
Title quality	$[-30, 25]$	Length, word count, gibberish detection, low-effort titles, structured prefixes, technical keywords
Description quality	$[-30, 35]$	Length, word count, gibberish, template-only structure, joke markers, multi-line formatting, lists
Content coherence	$[-20, 15]$	Overlap between title and description after stopword removal; penalties when the description is unrelated to the title
Style	$[-15, 20]$	Special-character ratio, repeated punctuation, repeated characters, excessive capitals, emoticons, sentence structure
Professionalism	$[-25, 20]$	Severe spam phrases, slang, insults, placeholders, begging patterns, newsletter-like non-issue text
Technical content	$[-10, 25]$	Software-engineering terms, code fragments, stack traces, file paths, version numbers, error markers

4.3. Local LLM-based scoring

Alongside OwnMetrics, we evaluated local small language models used in a zero-shot manner as issue-quality scorers. An initial screening phase discarded weaker candidates, after which the following four models were retained for detailed evaluation:

- Llama-3.1-8B-Instruct-Q4_1 [5],
- Mistral-7B-Instruct-v0.3.Q5_K_M [14],
- Phi-3.5-mini-instruct.Q8_0 [1],
- gemma-3-4b-it-qat-Q8_0 [8].

Each model was tested with temperatures 0.20, 0.30, 0.40, and 0.50. The models received only the issue title and description, and were instructed to return a JSON object of the form `{"p": <integer>}`. Early pilot experiments used a shorter prompt that emphasized output format but did not define the meaning of the score intervals precisely. This resulted in both more malformed outputs and lower accuracy. The final prompt therefore made the scoring semantics explicit.

This prompt design is important for two reasons. First, it reduces ambiguity between neighboring score regions. Second, it makes thresholding easier later, because the models are explicitly told what low, medium, and high scores represent.

Table 2. Score semantics communicated to the local LLMs.

Score band	Meaning given to the LLMs
0–20	Spam, advertisements, offensive content, empty reports, or gibberish
21–40	Not software-related, off-topic, or personal/social discussion
41–60	Vague or unclear issue with missing context
61–80	Acceptable issue that is understandable but lacks detail
81–100	High-quality issue with a clear problem statement, reproduction information, or a specific feature request

5. Experimental setup

An issue is classified as spam/noise when its validity score is below a threshold τ. We tested multiple thresholds and selected the best-performing value for each model configuration on the labeled benchmark. The best threshold was 75 for OwnMetrics and for most non-Llama configurations, whereas the best Meta-Llama-3.1-8B results were obtained at threshold 50. The full-benchmark results in Table 3 should therefore be interpreted as exploratory threshold-selection results rather than as unbiased estimates of generalization performance. To check the stability of this selection without additional model inference, we also performed a stratified five-fold threshold-selection validation over the already generated validity-score outputs: in each fold, the model configuration and threshold were selected on the training folds by accuracy, using F1 as a tie-breaker, and evaluated on the held-out fold.

The evaluation has three parts. First, we measure accuracy and F1 on the labeled benchmark. Second, we evaluate the selected configurations on the manually labeled 300-item Jira-based TAWOS validation set described above; this set is not used for threshold or model selection. Third, we examine deployment-oriented behavior on a 10,000-item sample drawn from the processed TAWOS issue corpus. The sample was created by the export script before validity filtering: issues were selected from the TAWOS `Issue` table using MySQL-level random ordering (`ORDER BY RAND()`) and the configured export limit was set to 10,000. The resulting CSV snapshot was then kept fixed and reused for all evaluated methods. No class labels, model scores, or threshold decisions were used when constructing this sample, and it was used only for robustness, score-distribution, agreement, spam-rate, and runtime analyses, not for training or threshold selection. Since runtime depends on the hardware and local software stack, the reported times should be interpreted comparatively rather than as universal constants.

For the broad comparison beyond the best-accuracy table, we use one representative configuration from each method family: OwnMetrics, Gemma-3-4B at temperature 0.50, Meta-Llama-3.1-8B at temperature 0.50, Mistral-7B at temperature 0.20, and Phi-3.5-mini at temperature 0.20. These configurations are used for the in-domain Jira/TAWOS validation and to compare deployment-oriented

behavior on the same fixed 10,000-item snapshot.

6. Results

6.1. Results on the labeled benchmark

Table 3 shows the best accuracy obtained by each model family over the tested temperature settings and thresholds in this exploratory threshold-selection benchmark.

Table 3. Best accuracy per model family on the labeled threshold-selection benchmark.

Model family	Temperature	Best accuracy	Best threshold
OwnMetrics	–	91.0%	75
Gemma-3-4B	0.20 / 0.30	89.2%	75
Mistral-7B-Instruct-v0.3	0.20	89.1%	75
Phi-3.5-mini-instruct	0.20	83.9%	75
Meta-Llama-3.1-8B-Instruct	0.20 / 0.50	66.2%	50

Figure 3 provides a visual summary of the same best-accuracy comparison across the evaluated model families.

OwnMetrics achieved the highest overall accuracy at 91.0%. Among the LLMs, Gemma-3-4B and Mistral-7B were the strongest, reaching 89.2% and 89.1%, respectively. Phi-3.5-mini was clearly usable but weaker, while Meta-Llama-3.1-8B lagged far behind the other methods.

Table 4 reports the stratified five-fold threshold-selection validation. The held-out results preserve the same ordering as the full-benchmark selection: OwnMetrics remains the strongest method, while Gemma-3-4B and Mistral-7B form the closest LLM baselines. The cross-validation also selected the same threshold pattern as the full benchmark in all folds: $\tau = 75$ for OwnMetrics, Gemma, Mistral, and Phi, and $\tau = 50$ for Meta-Llama.

The confusion-matrix values of the selected configurations are summarized in Table 5. OwnMetrics produced the strongest balance between false positives and false negatives. The selected Gemma and Mistral configurations were close to it, whereas Meta-Llama-3.1-8B missed a very large number of spam items.

6.2. OwnMetrics ablation

To better understand the contribution of the OwnMetrics components, we performed a small ablation study on the labeled benchmark. Since OwnMetrics is deterministic, this analysis does not require retraining or additional model inference. Removed components were excluded from the raw score, and the remaining component range was re-normalized to the 0–100 validity scale. For comparability

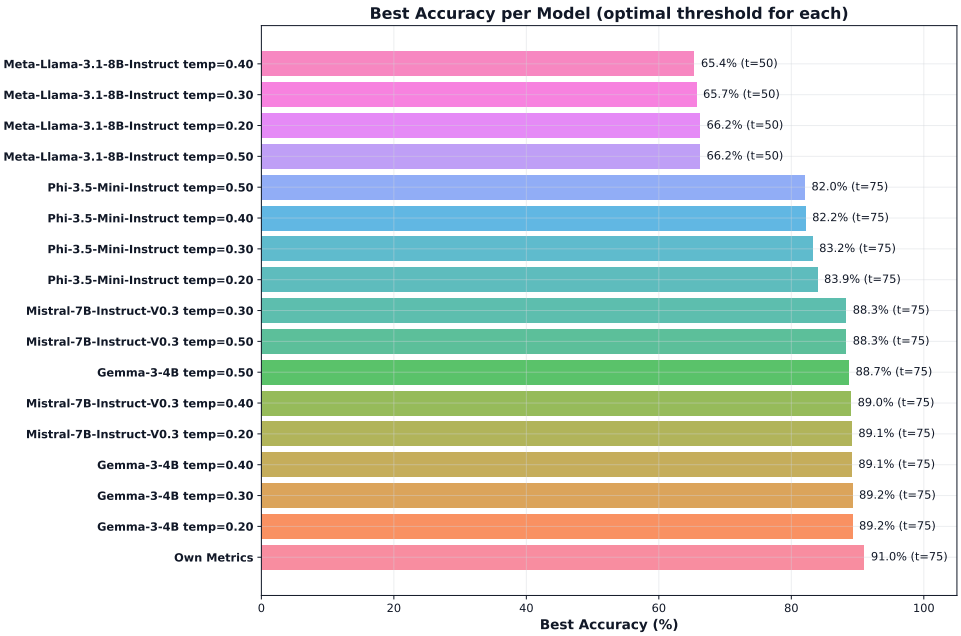

Figure 3. Best accuracy per model on the labeled threshold-selection benchmark.

Table 4. Stratified five-fold threshold-selection validation on the labeled benchmark. In each fold, the model configuration and threshold are selected on the training folds and evaluated on the held-out fold. Spam/noise is treated as the positive class. Accuracy and F1 are mean ± standard deviation; precision and recall are fold means. All values are percentages.

Model family	Accuracy	F1	Precision	Recall
OwnMetrics	91.0 ± 2.0	91.0 ± 2.1	92.1	90.0
Gemma-3-4B	88.8 ± 3.0	89.0 ± 2.8	89.1	89.0
Mistral-7B-Instruct-v0.3	88.8 ± 1.6	88.0 ± 1.6	96.3	81.1
Phi-3.5-mini-instruct	83.9 ± 2.1	83.9 ± 2.0	85.7	82.3
Meta-Llama-3.1-8B-Instruct	65.8 ± 2.3	49.4 ± 5.6	98.8	33.1

with the main filtering setting, all variants were evaluated using the fixed operational threshold $\tau = 75$.

The ablation results show that the early spam detector is the most influential component: removing it substantially reduces recall and F1. The technical-content and professionalism components also contribute to the final performance, while a variant using only title and description quality components is clearly insufficient.

Table 5. Confusion-matrix summary for selected model configurations on the labeled benchmark. Spam is treated as the positive class.

Method	TN	FP	FN	TP	Accuracy	F1
OwnMetrics	429	37	48	433	91.0%	91.1%
Gemma-3-4B ($t = 0.50$)	415	51	56	425	88.7%	88.8%
Mistral-7B ($t = 0.20$)	451	15	88	393	89.1%	88.4%
Phi-3.5-mini ($t = 0.20$)	399	67	85	396	83.9%	83.9%
Meta-Llama-3.1-8B ($t = 0.50$)	449	17	308	173	65.7%	51.6%

Table 6. Ablation analysis of OwnMetrics on the labeled benchmark at the fixed operational threshold $\tau = 75$.

Configuration	Accuracy	Precision	Recall	F1
Full OwnMetrics	91.0%	92.1%	90.0%	91.1%
Without early spam detector	79.7%	94.5%	63.8%	76.2%
Without technical-content component	90.1%	92.9%	87.1%	89.9%
Without professionalism component	89.7%	90.1%	89.4%	89.8%
Title and description components only	73.5%	94.6%	50.7%	66.0%

This suggests that the reported performance is not explained by a single length-based heuristic, but by the combination of early spam detection and multiple issue-quality signals.

6.3. In-domain Jira/TAWOS validation

The additional in-domain validation results are shown in Table 7. The set consists of 300 manually labeled Jira-based TAWOS issues. The selected thresholds and model configurations were fixed before this validation and were not tuned on the Jira/TAWOS set. Spam/noise is treated as the positive class. Since the set contains only 39 spam/noise cases and 261 non-spam cases, accuracy alone would be misleading; a majority-class baseline would already reach 87.0% accuracy while detecting no spam/noise issues. We therefore emphasize balanced accuracy, spam-class precision, recall, and F1. Invalid or unparsable outputs were treated as keep/non-spam decisions for the binary metrics and are also reported separately as failures.

The in-domain validation is consistent with the main benchmark conclusion at the level of a small Jira/TAWOS sanity check. OwnMetrics achieves the highest overall accuracy and the strongest spam-class F1, while producing no parsing failures. Gemma-3-4B obtains the highest spam recall, but it also produces more false positives. Mistral-7B remains close to OwnMetrics. Meta-Llama-3.1-8B has a seemingly high overall accuracy because most records are non-spam, but its bal-

Table 7. In-domain validation on 300 manually labeled Jira/TAWOS issues. The set contains 39 spam/noise and 261 non-spam issues. Thresholds and model configurations were selected before this validation and were not tuned on the Jira/TAWOS validation set. Spam/noise is the positive class, and all metric values are percentages except the confusion-matrix counts.

Method	τ	TP	FP	FN	TN	Acc.	Bal. acc.	Prec.	Rec.	F1	Fail.
OwnMetrics	75	33	28	6	233	88.7	86.9	54.1	84.6	66.0	0.0
Gemma-3-4B ($t = 0.50$)	75	36	48	3	213	83.0	87.0	42.9	92.3	58.5	0.0
Meta-Llama-3.1-8B ($t = 0.50$)	50	7	8	32	253	86.7	57.4	46.7	17.9	25.9	51.0
Mistral-7B ($t = 0.20$)	75	32	29	7	232	88.0	85.5	52.5	82.1	64.0	0.0
Phi-3.5-mini ($t = 0.20$)	75	34	48	5	213	82.3	84.4	41.5	87.2	56.2	1.7

anced accuracy and spam recall are low, and it is strongly affected by parsing failures.

6.4. Robustness on a 10,000-item sample

Accuracy alone is not sufficient for large-scale cleaning. Since the LLMs had to return a strict JSON format, we also measured output failures. OwnMetrics produced no failures at all. Gemma and Mistral were highly reliable, remaining around 0.1% failure. Phi-3.5-mini stayed below 1.2%. Meta-Llama-3.1-8B, however, failed on more than half of the 10,000-sample experiment, with failure rates between 54.0% and 59.2% depending on temperature.

Table 8 combines score statistics, failure rates, and runtime for the selected representative configurations. The median validity scores are consistently higher than the means, which indicates that most issues are rated as rather valid while a smaller set of heavily penalized spam items pulls the average downward. The LLMs also tend to have lower dispersion than OwnMetrics, suggesting more compressed internal scoring behavior.

Table 8. Summary statistics for selected representative configurations on a 10,000-item sample.

Method	Temp	Mean	Median	Std.	Min	Max	Failure	Time
OwnMetrics	-	78.8	85.0	21.1	5.0	96.0	0.0%	2.28 ms
Gemma-3-4B	0.50	78.6	85.0	13.4	0.0	100.0	0.1%	2.86 s
Meta-Llama-3.1-8B	0.50	79.8	81.0	19.5	0.0	100.0	54.0%	4.29 s
Mistral-7B	0.20	76.1	81.0	14.6	0.0	100.0	0.1%	5.77 s
Phi-3.5-mini	0.20	80.0	81.0	11.2	0.0	85.0	0.5%	4.78 s

6.5. Score distributions and spam agreement

The score-distribution statistics reveal an interesting trade-off. OwnMetrics is more spread out than the LLMs, which is consistent with its explicit penalties and rewards for many different textual phenomena. The LLMs compress more of their predictions into the upper mid-range, even when they separate spam from non-spam reasonably well. This can be useful if one wants stable relative ranking, but it may be less informative when strong score contrast is desired.

Pairwise agreement between the selected methods is also high. On the 10,000-item sample, the pairwise spam-decision agreement values lie between 0.83 and 0.87. This suggests that the models, despite their architectural differences, recognize a substantial common core of spam/noise cases. At the same time, the agreement is not perfect, which leaves room for ensemble strategies or manual review near the threshold boundary.

6.6. Spam detection rate and full-corpus filtering

At threshold 75, the selected configurations mark noticeably different proportions of the 10,000-item sample as spam. OwnMetrics flags 16.6% of items, Gemma around 19.2%, Mistral around 13.7–14.2%, Phi around 19.7–20.9%, and Meta-Llama only about 3.3–5.3%. This shows that even when two methods achieve similar benchmark accuracy, they may still define the operational spam boundary somewhat differently.

When the complete filtering pipeline was applied to the processed TAWOS issue corpus, 16.2% of records were flagged as spam-like, noisy, or otherwise unsuitable for further use. The resulting filtered corpus is intended to provide a cleaner basis for later extraction of statistically meaningful development-process properties and for realistic simulation studies.

7. Discussion and threats to validity

The empirical picture is consistent across the evaluation dimensions. OwnMetrics achieved the best exploratory benchmark accuracy, the strongest result in the five-fold threshold-selection validation, the best spam-class F1 on the small in-domain Jira-based TAWOS validation set, zero parsing failures, and by far the lowest runtime. The ablation study further indicates that this performance does not come from a single length-based heuristic: the early spam detector has the largest effect, but the technical-content and professionalism components also contribute. This suggests that for the present task, a hand-crafted scorer can outperform small local general-purpose LLMs when the target signal is dominated by surface-level and medium-depth textual regularities such as placeholders, promotional phrases, malformed content, missing coherence, and recognizable software-engineering vocabulary.

The LLMs nevertheless remain interesting baselines. Gemma-3-4B and Mistral-7B came surprisingly close to OwnMetrics in accuracy, and their pairwise agreement

with the rule-based method remained high. This indicates that local LLMs can be viable for issue validation even without task-specific training, provided that the prompt clearly defines the scoring semantics and the expected output format. However, the large gap in runtime and the non-negligible risk of malformed outputs remain practical disadvantages.

A first threat to validity comes from the labeled benchmark itself. The benchmark was built from GitHub issues, while the target corpus is Jira-based TAWOS data. Although the basic task – distinguishing software-related, meaningful issue descriptions from spam or noise – is similar across the two platforms, style differences, issue-management workflows, community norms, and moderation practices may not transfer perfectly. Consequently, the benchmark results should be treated as controlled evidence for spam/noise discrimination rather than as a direct estimate of in-domain TAWOS performance. The five-fold threshold-selection validation reduces the risk that the reported ranking is only a result of choosing a threshold on the full benchmark, but it does not remove this platform-shift limitation because the folds still come from the same GitHub-based benchmark. The additional 300-item Jira-based TAWOS validation set partially addresses this concern and provides direct in-domain evidence for the selected configurations. However, because it is small, imbalanced, and partly enriched with manually searched suspicious records, it should be interpreted as a diagnostic sanity check rather than as a full TAWOS gold standard or a prevalence estimate. A larger in-domain validation set would be needed for stronger external validation. A second threat is that only title and description were used. TAWOS also contains comments, change logs, issue links, users, versions, and sprint information [21]; these may provide useful extra evidence in ambiguous cases. A third threat is that runtime measurements are environment-specific. Finally, moderator-locked GitHub spam is a pragmatic but incomplete proxy for the broader notion of "noise": some low-quality issues are not spam, and some spam-like reports may escape moderation.

These threats also point to natural future work. The most promising extensions are to incorporate metadata beyond the title and description, to use hybrid decision rules where OwnMetrics handles obvious cases and an LLM reviews uncertain ones, and to calibrate the threshold based on the downstream task. For example, a training corpus for fine-tuning may justify a stricter threshold than a corpus intended for exploratory repository mining.

8. Conclusion

We presented a practical data-cleaning pipeline for the TAWOS issue database and focused on a validity-scoring stage that assigns a score from 0 to 100 to each issue text. The proposed OwnMetrics rule-based method combines an early spam detector with six interpretable scoring dimensions and achieved the strongest overall performance in our experiments. The added ablation analysis shows that the early spam detector is especially important, while the other scoring components also contribute to the final result. The small Jira-based TAWOS validation set provides

an in-domain sanity check consistent with the benchmark results. Among the local LLMs, Gemma-3-4B and Mistral-7B came closest in accuracy, but neither matched OwnMetrics in robustness or efficiency.

From a practical perspective, the main result is simple: for this type of large-scale issue cleaning, a domain-tailored deterministic scorer is already strong enough to serve as the backbone of an operational filtering pipeline. At the same time, the LLM experiments show that zero-shot local models are competitive enough to remain useful as baselines, secondary reviewers, or ensemble components. Applying the complete pipeline to the processed TAWOS corpus flagged a substantial share of records as potential noise, yielding a cleaner starting point for later educational, analytical, and modeling tasks.

References

[1] M. Abdin et al.: *Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone*, arXiv preprint arXiv:2404.14219 (2024), arXiv: 2404.14219 [cs.CL].

[2] G. Antoniol, K. Ayari, M. D. Penta, F. Khomh, Y.-G. Guéhéneuc: *Is It a Bug or an Enhancement? A Text-Based Approach to Classify Change Requests*, in: Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds, IBM Corp., 2008, pp. 304–318, doi: 10.1145/1463788.1463819.

[3] G. Colavito, F. Lanubile, N. Novielli: *Benchmarking Large Language Models for Automated Labeling: The Case of Issue Report Classification*, Information and Software Technology 184 (2025), p. 107758, doi: 10.1016/j.infsof.2025.107758.

[4] G. Colavito, F. Lanubile, N. Novielli, L. Quaranta: *Impact of Data Quality for Automatic Issue Classification Using Pre-Trained Language Models*, Journal of Systems and Software 210 (2024), p. 111838, doi: 10.1016/j.jss.2023.111838.

[5] A. Dubey et al.: *The Llama 3 Herd of Models*, arXiv preprint arXiv:2407.21783 (2024), arXiv: 2407.21783 [cs.LG].

[6] R. Ehsani, M. M. Imran, R. Zita, K. Damevski, P. Chatterjee: *Incivility in Open Source Projects: A Comprehensive Annotated Dataset of Locked GitHub Issue Threads*, in: Proceedings of the 21st IEEE/ACM International Conference on Mining Software Repositories, Association for Computing Machinery, 2024, pp. 515–519, doi: 10.1145/3643991.3644887.

[7] I. Ferreira, B. Adams, J. Cheng: *How Heated Is It?: Understanding GitHub Locked Issues*, in: Proceedings of the 19th International Conference on Mining Software Repositories, Association for Computing Machinery, 2022, pp. 309–320, doi: 10.1145/3524842.3527957.

[8] Gemma Team: *Gemma 3 Technical Report*, arXiv preprint arXiv:2503.19786 (2025), arXiv: 2503.19786 [cs.CL].

[9] GH Archive: *GH Archive*, 2026, url: https://www.gharchive.org/ (visited on 03/16/2026).

[10] GitHub: *REST API Endpoints for Issues*, 2026, url: https://docs.github.com/rest/issues/issues (visited on 03/16/2026).

[11] S. Herbold, A. Trautsch, F. Trautsch: *On the Feasibility of Automated Prediction of Bug and Non-Bug Issues*, Empirical Software Engineering 25.6 (2020), pp. 5333–5369, doi: 10.1007/s10664-020-09885-w.

[12] K. Herzig, S. Just, A. Zeller: *It's Not a Bug, It's a Feature: How Misclassification Impacts Bug Prediction*, in: Proceedings of the 2013 International Conference on Software Engineering, IEEE Press, 2013, pp. 392–401, doi: 10.1109/ICSE.2013.6606585.

[13] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, H. Wang: *Large Language Models for Software Engineering: A Systematic Literature Review*, ACM Transactions on Software Engineering and Methodology 33.8, 220 (2024), pp. 1–79, DOI: 10.1145/3695988.

[14] A. Q. Jiang et al.: *Mistral 7B*, arXiv preprint arXiv:2310.06825 (2023), arXiv: 2310.06825 [cs.CL].

[15] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, D. Damian: *An In-Depth Study of the Promises and Perils of Mining GitHub*, Empirical Software Engineering 21.5 (2016), pp. 2035–2071, DOI: 10.1007/s10664-015-9393-5.

[16] M. Laiq, F. Dobslaw: *Automatic Techniques for Issue Report Classification: A Systematic Mapping Study*, Automated Software Engineering 33, 72 (2026), DOI: 10.1007/s10515-026-00616-x.

[17] N. Nikeghbal, A. H. Kargaran, A. Heydarnoori: *GIRT-Model: Automated Generation of Issue Report Templates*, in: Proceedings of the 21st IEEE/ACM International Conference on Mining Software Repositories, Association for Computing Machinery, 2024, pp. 407–418, DOI: 10.1145/3643991.3644906.

[18] G. Rejithkumar, P. R. Anish, S. Ghaisas: *Text-To-Text Generation for Issue Report Classification*, in: Proceedings of the 3rd ACM/IEEE International Workshop on Natural Language-based Software Engineering, Association for Computing Machinery, 2024, pp. 53–56, DOI: 10.1145/3643787.3648042.

[19] M. Soltani, F. Hermans, T. Bäck: *The Significance of Bug Report Elements*, Empirical Software Engineering 25.6 (2020), pp. 5255–5294, DOI: 10.1007/s10664-020-09882-z.

[20] E. Sülün, M. Saçakçı, E. Tüzün: *An Empirical Analysis of Issue Templates Usage in Large-Scale Projects on GitHub*, ACM Transactions on Software Engineering and Methodology 33.5, 117 (2024), pp. 1–28, DOI: 10.1145/3643673.

[21] V. Tawosi, A. Al-Subaihin, R. Moussa, F. Sarro: *A Versatile Dataset of Agile Open Source Software Projects*, in: Proceedings of the 19th International Conference on Mining Software Repositories, New York, NY, USA: Association for Computing Machinery, 2022, pp. 707–711, DOI: 10.1145/3524842.3528029.

[22] F. Tu, F. Zhang: *Measuring the Quality of Issue Tracking Data*, in: Proceedings of the 6th Asia-Pacific Symposium on Internetware, Association for Computing Machinery, 2014, pp. 76–79, DOI: 10.1145/2677832.2677844.

[23] X. Wu, W. Zheng, X. Xia, D. Lo: *Data Quality Matters: A Case Study on Data Label Correctness for Security Bug Report Prediction*, IEEE Transactions on Software Engineering 48.7 (2022), pp. 2541–2556, DOI: 10.1109/TSE.2021.3063727.

[24] J. Zhang, X. Wang, D. Hao, B. Xie, L. Zhang, H. Mei: *A Survey on Bug-Report Analysis*, Science China Information Sciences 58.2 (2015), pp. 1–24, DOI: 10.1007/s11432-014-5241-2.

[25] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schröter, C. Weiss: *What Makes a Good Bug Report?*, IEEE Transactions on Software Engineering 36.5 (2010), pp. 618–643, DOI: 10.1109/TSE.2010.63.