Advancing LLM-Based Issue Report Classification with Explained Few-Shot Learning, Intent Extraction, Ensemble, and Summarization

GABRIELE DE VITO, Università degli Studi di Salerno, Italy
LUIGI LIBERO LUCIO STARACE, Università degli Studi di Napoli Federico II, Italy
FABIO PALOMBA, Università degli Studi di Salerno, Italy
SERGIO DI MARTINO, Università degli Studi di Napoli Federico II, Italy
FILOMENA FERRUCCI, Università degli Studi di Salerno, Italy

Effective software maintenance requires automated issue report classification due to increasing report volume and complexity. Although fine-tuned BERT models and Large Language Models (LLMs) have exhibited potential in this field, they face critical limitations in handling lengthy reports and ensuring classification consistency. This paper presents an LLM-based method for processing long reports and explores two classification perspectives, namely user intent understanding and example-based decision-making. On this basis, we propose three LLM-based methods: (1) Intent Extraction and Classification, which identifies and classifies user intent from issue reports; (2) Ensemble Classification, which enhances the intent-based method through majority voting; and (3) Explained Few-Shot Learning, which implements the example-based strategy with transparent rationales. We compare these methods against 3 baselines: a RoBERTa-based model, SETFIT, and a previous LLM-based method, using GPT-4o, GPT-3.5-turbo, and Qwen 2.5-32B, through an extensive evaluation that comprises consistency analysis, ablation studies, and an analysis of misclassification patterns. The results show that GPT-4 outperforms the state-of-the-art by 5–8% and performs well across all methods. Furthermore, the results show that Qwen-2.5 performs better than the larger GPT-3.5-turbo, suggesting that multiple factors beyond parameter count–including architectural design, training data composition, and optimization strategies–influence classification performance. The analysis creates a taxonomy of classification challenges and reveals important findings about the pros and cons of each approach. We also introduced innovative ensemble techniques based on LLM perplexity and adaptive strategies capable of selecting the most effective LLM and proposed classification method under specific privacy constraints.

CCS Concepts: • **Software and its engineering** → *Software maintenance tools*; **Maintaining software**; **Software evolution**; • **Information systems** → *Decision support systems*.

Additional Key Words and Phrases: Issue Report Classification, Software Maintenance, Large Language Models

1 Introduction

In software development and maintenance, issue reports serve as essential communication mechanisms between stakeholders and development teams. Managed through specialized tracking tools [5], these reports formally document change requests, bugs, and other concerns encountered during

Authors' Contact Information: Gabriele De Vito, gadevito@unisa.it, Università degli Studi di Salerno, Salerno, Italy; Luigi Libero Lucio Starace, luigiliberolucio.starace@unina.it, Università degli Studi di Napoli Federico II, Napoli, Italy; Fabio Palomba, Università degli Studi di Salerno, Salerno, Italy, fpalomba@unisa.it; Sergio di Martino, sergio.dimartino@unina.it, Università degli Studi di Napoli Federico II, Napoli, Italy; Filomena Ferrucci, Università degli Studi di Salerno, Salerno, Italy, fferrucci@unisa.it.

ACM 1557-7392/2025/8-ART111
https://doi.org/XXXXXXX.XXXXXXX

software development and use. Throughout the project lifecycle, developers can systematically address user needs and software defects by capturing from issue reports crucial elements such as the problem's nature, reproduction steps, expected vs. actual behavior, and supporting evidence like screenshots or error logs [24]. Therefore, software maintenance and bug tracking depend on these reports being correctly classified, as they help teams in priority management [25], and thus enables proper resource allocation and timely issue resolution. Although stakeholders often provide initial classifications when submitting reports, these are often inaccurate [16, 21], and relying solely on manual reclassification creates processing bottlenecks. Given the high volume and complexity of issue reports, automating the classification process has become increasingly important. Various approaches have been proposed in the literature, including machine-learning techniques [3], text mining combined with data mining [49], and deep-learning models such as fastText [24]. Although these techniques have shown promise, they often face limitations in generalizability, reliance on manual data classification, and handling diverse linguistic patterns.

The advent of Large Language Models (LLMs) has shown potential in addressing the issue report classification problem [1, 14, 22]. Colavito et al. [14–16] demonstrated that GPT-like LLMs could achieve performance comparable to state-of-the-art BERT-like models without the need of fine-tuning. However, their work highlighted two critical limitations: handling long issue bodies effectively – as recent research [2, 43] demonstrates that LLM performance degrades as context size increases, even when content fits within technical context limits – and managing the non-deterministic behavior leading to inconsistent classifications. To address these challenges, we introduce an LLM-based solution that effectively handles long reports. The method adapts the MapReduce paradigm to process lengthy issue descriptions by dividing them into manageable chunks, processing each independently, and consolidating the results.

Building on this foundation, we examined two complementary perspectives to issue classification. The first one focuses on understanding the user's intent when creating the issue, using this intent as the basis for classification (i.e., "*what is the user trying to achieve?*"). The second perspective treats classification as a decision-making process based on explicit examples and criteria, similar to how a human would follow an informed checklist (i.e., like a triager consulting precedents with explicit reasons). In other words, human experts do not classify issues based on memorized patterns alone, but by examining examples and explicitly stating the motivations why each case belongs to a specific category [40]. These fundamental perspectives inform our three classification methods: "Intent Extraction and Classification" and "Ensemble Classification," which implement the intent-based approach (with the latter also addressing non-determinism). In contrast, "Explained Few-Shot Learning" implements the example-based approach. This dual strategy allows us to better understand LLMs' strengths and limitations in classification tasks.

To assess the methods, we have performed an empirical evaluation addressing the following research questions:

> **RQ1:** *How do the proposed LLM-based methods compare to existing approaches in terms of effectiveness and operational characteristics?*

This question aims to evaluate the effectiveness of the proposed methods against established baselines across multiple LLMs (GPT-4o, GPT-3.5, and Qwen 2.5) to assess whether they provide a tangible improvement in classification performance. Moreover, this question aims to evaluate operational characteristics of the LLM-based methods, including computational costs, handling of long texts, perplexity (i.e., model uncertainty in predictions), and parameter sensitivity across different LLMs.

RQ2: *To what extent are the classification results of the proposed methods consistent?*

This question investigates the non-deterministic behavior of LLMs and aims to evaluate the consistency of the classification outcomes. By examining the variability in classifications for the same issue report across multiple evaluations, we can determine the reliability of our approaches.

RQ3: *What are the predominant error patterns and behavioral dynamics exhibited by the proposed LLM-based classification methods?*

This question investigates the error patterns and behavioral dynamics of the proposed methods in different LLMs, examining how errors are distributed, corrected, or consolidated to identify systematic biases and method-specific failure modes.

The methodological framework used in our study is depicted in Figure 1.

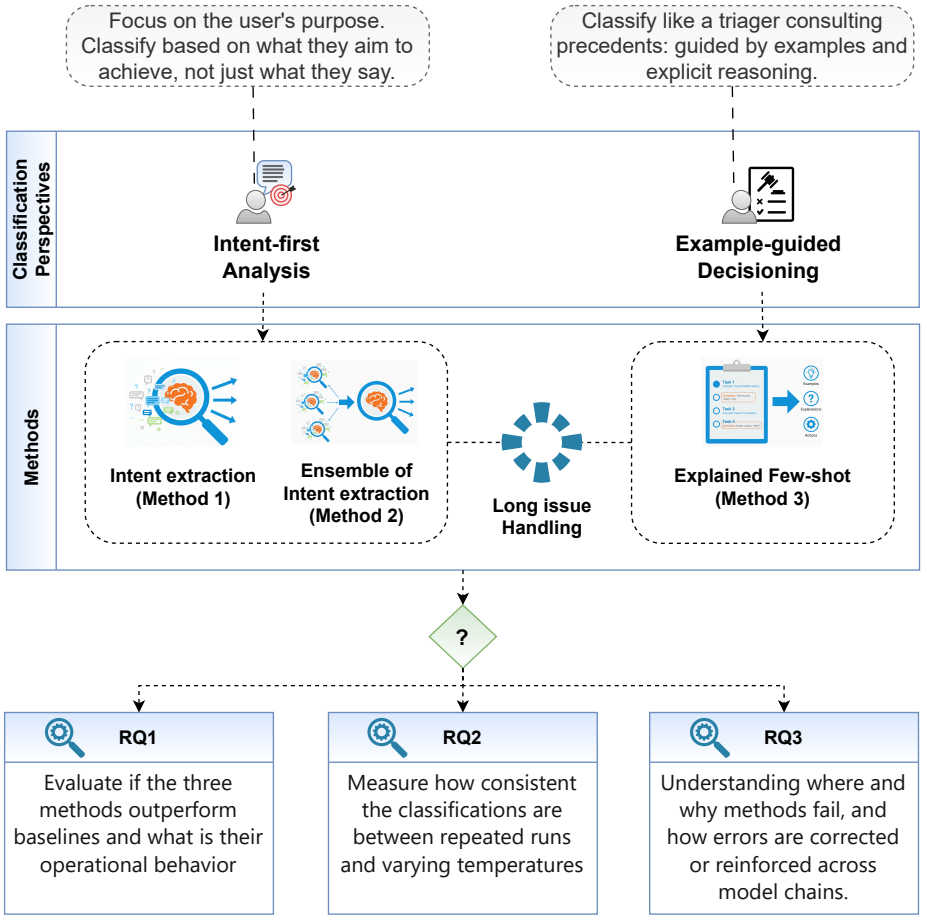

Fig. 1. Overall Methodology used in the study.

The empirical evaluation demonstrates that all proposed methods outperform existing approaches across three LLMs, achieving 5-8% F1-score improvements. Beyond performance gains, the analysis reveals counterintuitive behaviors: ensemble methods can consolidate errors rather than

correct, highlighting the limitations of majority voting when base models share systematic biases. We demonstrate that the MapReduce approach consistently outperforms both truncation and full-context ingestion for long issue reports, offering superior classification effectiveness while maintaining manageable inference costs. Additionally, we identify class-level perplexity as a reliable predictor of classification confidence, enabling more adaptive and trustworthy deployment strategies. The analysis also establishes a taxonomy of misclassification causes and reveals that different model architectures exhibit complementary failure patterns, suggesting opportunities for more sophisticated ensemble strategies.

All in all, the contribution of our work is fourfold:

(1) We introduce three LLM-based methods that mimic expert decision processes, surpass the state of the art, and aim to address key SE challenges in report classification: consistency, variable quality, linguistic heterogeneity, and lengthy descriptions.

(2) We provide the first analysis of LLM behavior in issue classification, revealing key findings about ensemble pros and cons, temperature sensitivity across models, and how perplexity metrics can predict classification confidence.

(3) We establish a taxonomy of misclassification causes specific to software issues (class overlap, ambiguity, complexity, implicit intent) and identify systematic error patterns across LLM architectures, providing actionable insights for industrial deployment in privacy-constrained and privacy-unconstrained settings.

(4) Informed by the behaviour analysis and the taxonomy of misclassification causes, we propose: (i) novel perplexity-based ensemble methods that leverage model-internal confidence signals; and (ii) adaptive routing strategies designed for privacy-sensitive and privacy-unconstrained scenarios, designed to maximize classification performance while controlling costs under constrained conditions.

(5) We provide new manually curated datasets of issue reports to the research community, complementing Colavito et al.'s work [15] and following their verification procedure. The dataset is available in the replication package at [18].

Structure of the paper. Section 2 provides relevant background notions and discusses related work on issue report classification. Section 3 illustrates the proposed methods. Section 4 outlines the empirical evaluation setup. Section 5 presents and discusses the results, while Section 6 covers the threats to the validity of our study. Last, Section 7 concludes the paper.

2 Background and Related Work

In the following, we present the background necessary to comprehend our work and its relevant context. First, we briefly introduce LLMs, used to create the core components of the suggested approach. Then, we provide an overview of existing approaches to issue report classification and the motivation behind our work.

2.1 Large Language Models

LLMs have transformed the field of Natural Language Processing (NLP) by using the Transformer architecture, which features self-attention mechanisms, as introduced by Vaswani et al. [42]. Prominent examples of these models include commercial offerings like GPT-3 [7] and GPT-4 [33], as well as open-source models such as BERT [19], FLAN-T5 [11], LLama [41], BLOOM [38], and GLM [47]. LLMs are trained on vast text datasets and often comprise hundreds of billions of parameters [9, 28, 48]. The initial "pre-training" phase is computationally demanding but crucial for enabling these models to excel at various NLP tasks, such as translation and summarization [9,

28, 48]. After pre-training, LLMs can be further specialized through a fine-tuning process, which involves using smaller datasets to customize the models for specific NLP tasks, such as question-answering or domain-specific applications. Several emergent abilities have been identified in LLMs, including "In-context learning", "Instruction following", and "Step-by-step reasoning". "In-context learning" allows models to perform tasks based on examples without additional training. "Instruction following" enables the models to execute tasks based solely on given instructions, while "Step-by-step reasoning" helps solve complex problems through chain-of-thought prompting [44]. However, despite their impressive capabilities, LLMs face several challenges that warrant attention. Their inability to handle lengthy texts due to context window limits is a major problem that can lead to information loss when processing lengthy text inputs. Furthermore, LLMs' non-deterministic behavior, especially when it comes to classification tasks, can introduce inconsistencies that affect automated systems' dependability.

2.2 Issue Report Classification

Traditional machine learning methods were the mainstay of early automated approaches to issue report classification. Antoniol et al. [3] utilized alternating decision trees, naïve Bayes classifiers, and logistic regression, to automatically categorize issues in bug tracking systems as bugs (for corrective maintenance) or non-bugs. Their technique achieved an accuracy of 77% to 82%, indicating the potential for automated issue classification. Similarly, Zhou et al. [49] tackled the same bug/non-bug issue report classification problem and proposed a multi-stage approach that combines text mining and data mining techniques to further improve classification performance. Their approach, evaluated on ten large open-source projects, outperformed baseline methods and individual classifiers and demonstrated significant improvement in classification performance w.r.t. the approach proposed by Antoniol et al.

Kallis et al. [24] introduced Ticket Tagger, a GitHub app that automates issue labeling using a fastText machine-learning model. The model classifies issues such as bug reports, enhancements, or questions based on their titles and descriptions. Evaluation on a dataset of 30,000 GitHub issues showed a high F1-score across categories (> 80%). However, challenges were noted with false positives in questions and false negatives in feature requests, suggesting the need for improved handling of diverse linguistic patterns in issue descriptions.

Recent studies have also explored the potential of LLMs for issue report classification, leveraging their ability to process natural language [1, 4, 14, 16, 22, 26]. The organizers of the Issue Report Classification Challenge within the 2nd International Workshop on Natural Language-based Software Engineering (NLBSE' 23) introduced a RoBERTa-based model, using a crowd-sourced dataset, that they distributed in the scope of the challenge [26]. The model still represents the state-of-the-art and achieved an F1-micro score of 0.8890. Nonetheless, Colavito et al. noted that crowd-sourced datasets often contain inconsistent and noisy labels, which can affect the performance of models [14, 15]. To mitigate this issue, they used a manually verified subset of the NLBSE'23 Issue Report Classification Challenge dataset [26], that they distributed in their previous work [15], and employed SETFIT, a few-shot learning approach for fine-tuning transformer-based models, to train a classifier. Their model achieved an F1-micro score of 0.8321, comparable with the state-of-the-art [26].

In another study, Colavito et al. also suggested using GPT-like LLMs, without fine-tuning, to automate issue labeling in software projects [16]. Their study showed that LLMs, specifically GPT-3.5-turbo, using few-shot learning, can achieve performance similar to top BERT-like models even without fine-tuning, achieving a promising micro average F1-score of 0.8155. Further advancing the use of LLMs for Issue Report Classification, recent studies presented at the 2024 edition of the NLBSE Tool Competition [27] have investigated approaches based on fine-tuning LLMs [4, 36]. More in detail, [36] proposed fine-tuning a single instance of the Text-to-Text Transfer Transformer

(T5) LLM for the classification task using issue reports from different repositories. The approach presented in [4], on the other hand, is based on fine-tuning repository-specific versions of GPT-3.5-turbo. The fine-tuning process involved simulating conversational interactions, where the title and description of the issue reports were provided as input, and the actual label was used as the expected output. Both approaches were evaluated, within the context of the NLBSE'24 tool competition, on issue reports from five well-known software repositories. They showed promising results (0.83 weighted-averaged F1 on the considered data) that may be attributed to the fine-tuned model's ability to learn repository-specific concepts, technologies, and domain-related topics characterizing the considered repositories. However, fine-tuning LLMs is a computationally expensive task, which may limit the applicability of these approaches to smaller or less-resourced software repositories.

While recent studies have shown promising results using LLMs for issue report classification, key challenges persist in handling lengthy reports and ensuring consistent classifications. Our work builds upon these advances while addressing their limitations through novel approaches that we present in the following section.

3 Proposed Methods

In the following section, we present the three LLM-based methods proposed for issue report classification, which derive from two distinct conceptual perspectives. Understanding the purpose of the user's issue report creation is the main objective of the first perspective. Following this viewpoint, we developed the "*Intent Extraction and Classification*" method (Method 1, see Section 3.2), which uses an LLM to ascertain the underlying intent before labeling. The "*Ensemble Classification*" method (Method 2, see Section 3.3) extends this strategy by executing several separate instances of the Intent Extraction pipeline and combining their predictions via a voting mechanism to overcome the problem of non-determinism. The second perspective considers classification as a decision-making process based on clear motivations and examples. The "*Explained Few-Shot Learning*" method (Method 3, see Section 3.4) employs this viewpoint by using few-shot examples to guide the model in distinguishing between class differences through explicit reasoning and explanations. To support these methods, we adopt a specific strategy (Section 3.1) to handle lengthy issue reports by dividing them into manageable portions, processing each separately, and consolidating results. We now describe the approach to handling lengthy issue reports, followed by detailed descriptions of the three methods.

3.1 Handling Lengthy Issue Descriptions

The length of issue descriptions can vary greatly, which makes it difficult to work with LLMs because of the limited amount of context they can process. Although long issues in the NLBSE'23 dataset are infrequent (<5%), several studies [6, 8, 10, 39] revealed that, in software practice and industry, developers require high-quality issue reports that include detailed descriptions (crash details, reproduction steps, and stack traces, for effective classification, debugging, and resolution), which typically lead to verbose and longer descriptions. With medium-to-large-scale projects potentially producing over 30,000 issues a year, this requirement becomes even more crucial, because developer productivity and issue resolution efficiency can be disproportionately impacted in such situations by even a small percentage of lengthy, complex reports, such as the top 5% [39].

While modern LLMs support extended context windows (GPT-3.5 supports 16K tokens and GPT-4o supports 128K tokens), the choice to implement a MapReduce-based approach for long-text issues is based on recent empirical evidence about LLM performance degradation with extended contexts, as well as practical constraints. Colavito et al. [16] demonstrated in their work about issue classification using GPT-3.5, that including few-shot examples necessitated truncating issue descriptions, even with a context size of 16k tokens. More importantly, recent research has revealed

that extending context windows does not necessarily improve, but can degrade, LLM performance. Wang et al. [43] demonstrated that LLM accuracy significantly decreases as context length increases, with GPT-4 Turbo's accuracy dropping from approximately 18% at 4K tokens to only 3-5% at 16K tokens in sorting tasks. Similarly, An et al. [2] proved that models' ability to follow instructions deteriorates drastically in lengthy contexts, especially for query-based tasks, where LLMs find it difficult to understand long instructions.

Colavito et al. [16] addressed this challenge by cutting the issue description to a certain number of tokens to fit the constraints of GPT-3.5. Likewise, the RoBERTA-based approach, which represents the state of the art in issue classification, truncates the issue description to 512 tokens. However, this approach may lead to the loss of crucial information throughout the issue description, potentially resulting in omitted important details such as specific examples, context, or nuances within the text.

To overcome this limitation, we adopt an adapted version of the MapReduce paradigm [32] to process long texts. The approach is based on three main phases, as highlighted in Figure 2: (1) chunk division, (2) parallel processing to extract relevant information from each chunk, and (3) consolidation to provide the final response.

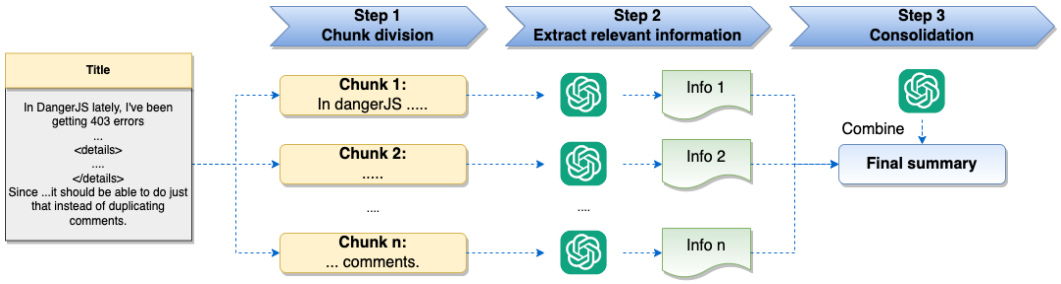

Fig. 2. Approach for Dealing with Long Text.

In the first phase, the issue description is divided into smaller chunks, which the LLM can manage. The chunk division is performed such that each chunk has a maximum size of *window size*, which is a configurable parameter, ensuring that the sum of tokens across all chunks equals the total number of tokens in the issue description, i.e., $\sum_{i=1}^{n} \text{tokens}(\text{chunk}_i) = \text{tokens}(\text{issue})$.

We decided to set *window size* to 4,000 tokens based on converging empirical evidence from recent research [2, 43]. The findings from Wang et al. and An et al. show that the challenge lies in maintaining quality as input length increases over 4k tokens, rather than simply fitting content within technical limits. Therefore, the 4,000-token threshold represents a pragmatic balance, because it provides sufficient context for semantic understanding while keeping LLMs within their optimal performance range and ensuring compatibility across different architectures.

In the second phase, each chunk is processed in parallel to extract relevant information using the LLM with a specific prompt, the "Extract Prompt." The "Extract Prompt" is designed to instruct the LLM to extract only relevant information from each segment or simply indicate «NONE» if there is no relevant information. This prompt is customized based on specific needs, for example, to identify the user's intent or to summarize the text of the issue, as we will see in the following sections.

Finally, in the Consolidation phase, we evaluate whether the total token count of all extracted information from the previous step exceeds the predefined *window size*. If it does, we create sublists of the extracted content where each sublist fits within the *window size* limit. Each sublist is then processed using the "Reduce Prompt" (reported in Figure 3) to generate summaries. If the combined

summaries still exceed the context window, the summarization process is recursively applied to the summaries themselves until the total content fits within the window size. Once the content is appropriately sized, a final extraction step is performed using the "Extract Prompt" to ensure the consolidated information maintains relevance and coherence. This reduction process continues until the final synthesized text fits within *window size* and is thus suitable for input into the classification methods.

Reduce Prompt

Your task is to shorten the following texts up to {tokens} tokens.
I'll provide a list of texts which relates to the same topic, and you will answer only with a shorter version that combines the texts.

Texts:
{texts}

Fig. 3. Reduce Prompt.

We note that the above-described MapReduce approach is applied only when issues exceed the specified *window size*. In the other cases, the relevant prompt can be directly applied to the entire description without dividing it into smaller parts.

3.2 Method 1: Intent Extraction and Classification

The "Intent Extraction and Classification" method consists of two main components (see Figure 4): 1) the Intent Extractor and 2) the Classifier. It uses the LLM to identify the user's intent behind reporting an issue and subsequently classifies the issue based on this intent.

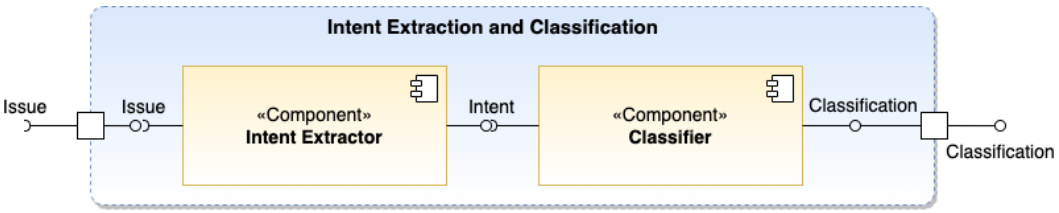

Fig. 4. Intent Extraction and Classification: Components.

For designing prompts to guide LLMs to mimic human workflow via prompting, we apply several key principles. First, we employ the Persona pattern [44], assigning the LLM the role of "expert issue classifier in the field of software engineering", allowing the model to produce more authoritative and pertinent responses. Second, we use a hierarchical instruction structure (Figure 6), where the model is guided by explicit conditional rules (e.g., "If the user is only reporting bugs... the classification must be 'Bug'"), with the aim of managing class overlap through clear priority rules when information that may concern multiple classes is present in the issue report. Third, to direct the model's attention away from superficial technical details and toward user intent, the intent-focused extraction prompt (Figure 5) contains specific linguistic cues.

The Intent Extractor adopts the approach described in section 3.1 to handle lenghty issue descriptions. Specifically, it uses the "Extract Prompt" version illustrated in Figure 5 to instruct the LLM to identify the user's intent from a specific text.

Extract Prompt: Intent Extraction.

You are an expert issue classifier in the field of software engineering.
The possible classifications are the following:
- Bug: {description of the bug classification}.
- Feature: {description of the feature classification}
- Question: {description of the question classification}
- Documentation: {description of the documentation classification}
- Unknown: for other cases.

You are tasked to extract from the partial text of a very long issue only the most relevant information for the issue. If the text is irrelevant, simply report "«NONE»".

Ensure the information you provide directly reflects the user's intent and clearly categorizes the issue to assist your colleagues in classifying the issue. For example, If the issue involves unexpected behavior, malfunctions, or errors but there is an explicit request or question for assistance, start your summary with: "The user is asking questions, feedback, explanations, seeking assistance or support...." For other bugs, start your summary with: "The user is reporting a bug where...." If the issue involves documentation, instructions, or help, start your summary with: "The user is requesting additions or improvements to the documentation by....".
If the issue is related to multiple classes, make sure that the summary reflects this aspect.

Answer only with the summary, not the classification.
issue title: {title}
issue body: {body}
Summary of the issue:

Fig. 5. Extract Prompt for Method 1.

The Classifier takes the text representing the user's intent obtained from the Intent Extractor as input and classifies it using the LLM instructed with the prompt in Figure 6.

3.3 Method 2: Ensemble Classification

The "Ensemble Classification" method is designed to enhance the robustness and reliability of the "Intent Extraction and Classification" method previously described (see 3.2). Indeed, the potential non-determinism in LLMs can lead to inconsistencies in intent extraction and subsequent classification of issues. To mitigate this, we employ an ensemble of multiple instances of the "Intent Extraction and Classification" pipeline. By aggregating the outputs from multiple instances, we aim to achieve a consensus that is more reliable and representative of the true intent behind the reported issue.

The ensemble method involves several key steps (see Figure 7).

We instantiate five parallel instances of the "Intent Extraction and Classification" pipeline. Each instance independently processes the same issue, extracting the user's intent and classifying the issue based on this intent. We note that, since each instance uses the intent extraction pipeline, lengthy issues are handled by the MapReduce-based approach described in section 3.1.

We choose to have five instances in the ensemble because there are four different classification categories. Indeed, despite possible outliers or residual non-determinism in the classification process, this configuration reduces the possibility of ties and increases classification reliability by guaranteeing a clear majority vote (at least three out of five) in the majority of cases. In

Method 1: Classification Prompt

You are an expert issue classifier in the field of software engineering. The possible classifications are the following:
- Bug: {description of the bug classification}.
- Feature: {description of the feature classification}
- Question: {description of the question classification}
- Documentation: {description of the documentation classification}
- Unknown: for other cases.

Instructions:
- If the user is only reporting bugs or errors, without seeking assistance, questions, feedback, explanations, or mentioning documentation, help, or instructions, the classification must be "Bug".
- If the user is explicitly asking questions, feedback, explanations, seeking assistance or support, the classification must be "Question", even if the user mentions bugs, errors, documentation, or other related issues.
- If the issue regards the documentation, help, or instructions, the classification must be "Documentation". This includes reporting missing information, errors, or suggesting updates to the documentation.
- If the user is requesting or suggesting a feature, the classification must be "Feature".
- In other cases, the classification must be "Unknown".

Examples:
{A Few examples}

I'll provide the body of the issue, and you will answer only with the classification.

issue body: {intent}
Classification:

Fig. 6. Prompt for classification in Method 1.

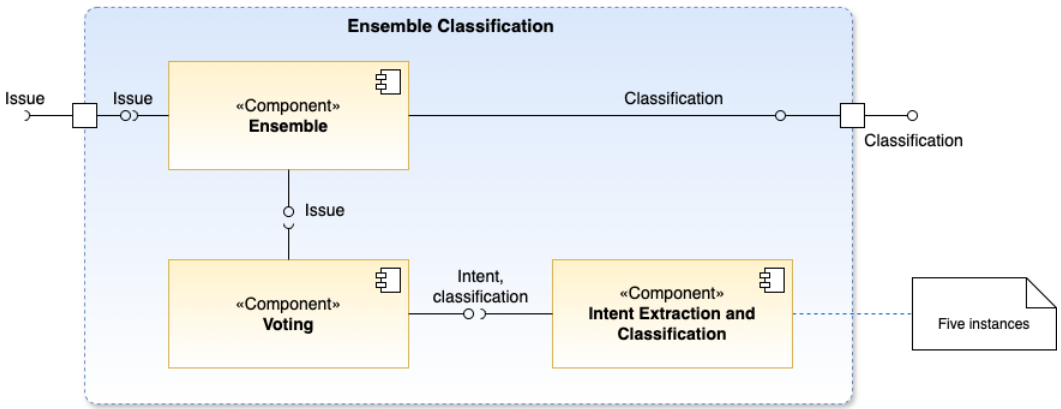

Fig. 7. Ensemble Classification: Components.

our implementation, tie scenarios (e.g., 2+2+1 distributions) are resolved by selecting the first

occurrence of the most frequent classification. More specifically, the most frequent result is chosen after counting occurrences across all instances and promoted to final classification.

3.4 Method 3: Explained Few-Shot Learning

Building upon Colavito et al.'s work [16], we enhance their few-shot learning approach by addressing the challenge of handling long issues and incorporating explicit reasoning. Our method combines few-shot learning with co-prompt engineering [17], a process that leverages GPT-4o to collaboratively generate a set of high-quality, synthetic few-shot examples, designed to guide the model in discriminating subtle differences between classes. The method has two main components (see Figure 8): the Summarizer and the Explained Few-Shot Classifier.

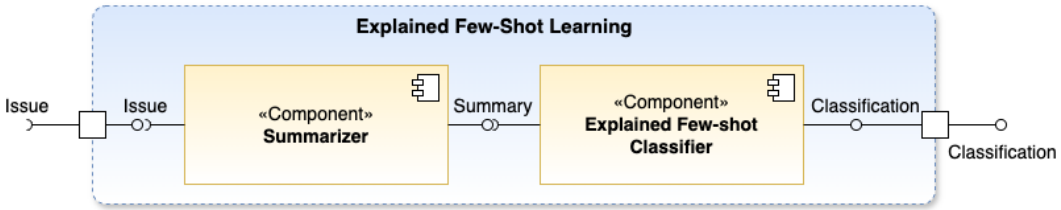

Fig. 8. Explained Few-Shot Learning: Components.

The Summarizer condenses the issue report if it is too long to fit the LLM's context. Then the Few-Shot Classifier classifies the resulting text.

3.4.1 Summarizer. The "Summarizer" component exploits the approach described in Section 3.1. Specifically, it uses the "Extract Prompt" version illustrated in Figure 9 to instruct the LLM to summarize the most relevant information from a specific text. However, if the issue description does not exceed the *window size*, it returns the original issue description.

> **Extract Prompt: Summarization.**
>
> The possible classifications are the following: Bug, Feature, Question, Documentation, Unknown.
>
> If the title and the text do not contain relevant information to classify the issue, simply report "«NONE»". You are tasked to extract from the title and the partial text of a very long issue only the most relevant information for the issue. If the text is irrelevant, simply report "«NONE»". When summarizing, ensure to include key details from both the title and the body of the issue. Answer only with the summary, not the classification.
>
> issue title: {title}
> issue body: {body}
> Summary of the issue:

Fig. 9. Extract Prompt for Method 3.

3.4.2 Explained Few-Shot Classifier. The "Explained Few-Shot Classifier" extends traditional few-shot learning.

Rather than simply providing examples, we accompany each example with an explicit explanation demonstrating why a particular classification decision was made. This modus operandi is similar

to how human experts make decisions, classifying objects based on explicit reasoning and learned patterns. In fact, human experts typically use explicit reasoning and mental checklists to categorize problems, using a variety of examples and concise justifications to help them make consistent and transparent decisions [40].

The development of the few-shot examples involved multiple phases. Initially, we randomly selected a stratified sample of 15 issues for each class from the training set provided by Colavito et al. [16], totaling 60 issues. We then utilized GPT-4o to generate, for each sample, a similar issue along with an explicit classification rationale, in a co-prompt engineering process [17]. We chose to generate synthetic examples based on real issues from Colavito et al. [16] rather than using unsupervised generation to ensure domain authenticity while increasing dataset diversity and coverage of edge cases [30, 37]. Two authors independently assessed the quality of these generated examples using a 5-point clarity scale (1=ambiguous, 5=very clear).

The clarity assessment took into account specific elements such as ambiguous terms without antecedents (e.g., "the previous version" without identifying which), unclear references (e.g., "it" or "this thing"), and lack of logical connections between the report content and the classification rationale. Examples with scores of 5 (very clear) had none of these instances, 4 (clear) had at most one instance, 3 (moderately clear) had two to three instances, 2 (unclear) had four to five instances, and 1 (very unclear) had more than five instances.

After this initial filtering, only examples with a score of ≥ 4 were kept, yielding 50 high-quality examples, with a significant evaluators' inter-rater reliability (Cohen's kappa = 0.78) and disagreements settled through discussion until agreement was reached. In order to further refine the selection, we removed redundant examples by calculating the cosine similarity between sentence embeddings (using OpenAI's text-embedding-3-small), pruning examples with similarity greater than 0.85 within categories, and retaining only instances with the highest rating. In the final example distribution, comprising 33 exemplars (2 Bug reports, 12 Questions, 7 Feature requests, 12 Documentation issues), categories with more linguistic diversity (Questions and Documentation) were given priority, and fewer examples were given to Bug reports, which showed more consistent linguistic patterns. We then designed the classification prompt following the approach of De Vito et al. [17], incorporating the 33 selected examples. As preliminary testing before our main experiment, we evaluated the classification prompt on Colavito et al.'s training dataset (separate from our test datasets), achieving F1 scores of 0.93 and 0.96 with GPT-3.5 and GPT-4o, respectively. Notably, this training set has no overlap with the datasets used in the empirical evaluation (see Section 4.4).

Figure 10 shows the final prompt structure that incorporates examples and reasoning guidelines.

3.4.3 Design Principles. The few-shot prompt (Figure 10) combines several design principles that enhance classification performance. First of all, we used the Persona pattern, as in Section 3.2. Second, as we have seen in Section 3.4.2, the approach is based on well-rounded and diversified examples. This strategic imbalance reflects the varying complexity of linguistic patterns across different issue types and aims at maximizing the information density of the few-shot prompt while remaining within context length constraints. Third, the integration of explicit reasoning through explanations represents a key innovation in our methodology. Each example includes both the classification and the rationale behind the decision, such as "The user classified the issue as a 'bug' because the nested button's inability to receive touch events... is an unintended behavior." This explanatory component guides the model's reasoning process and helps establish clear decision boundaries between classes.

Few-Shot Learning with Explanation Prompt.

You are an expert at classifying issues into one of the following categories: Bug, Feature, Question, Documentation, or Unknown. Below are some examples of issues similar to the new issue, along with their classifications. Use these examples to accurately classify the new issue provided. Focus on the explanations to understand how to classify the new issue.

Examples:
1. **issue:** "With 2 nested buttons, if the outside one has the prop 'enabled=false', the inside one does not receive touch events. Tested on iOS, not sure about Android."
Classification: Bug
Explanation: "The user classified the issue as a "bug" because the nested button's inability to receive touch events when the outer button is disabled is an unintended behavior that needs to be fixed."

[*other examples omitted for the sake of conciseness*]

33. **Issue:** "I'm not clear if this is a React or Redux issue. The state doesn't update correctly when dispatching an action."
Classification: Question
Explanation: The user is seeking clarification on whether the issue is related to React or Redux, indicating they need guidance to understand the source of the problem.

New Issue:
Issue: "{issue}"
Classification:

Follow the examples and answer only with the classification.

Fig. 10. Few-Shot Learning with Explanation Prompt.

4 Empirical Evaluation

In this section, we detail the research methodology and experimental design used to evaluate the three proposed methods.

In the remainder of the paper, for the sake of conciseness, we refer to "*Intent Extraction and Classification*" as Method 1, "*Ensemble Classification*" as Method 2, and "*Explained Few-Shot Learning*" as Method 3.

4.1 Dataset Selection

In the analyses, we utilized data extracted from the well-known NLBSE'23 Issue Report Classification Challenge dataset [26]. This dataset includes 1.4 million publicly-available issue reports, collected from GitHub repositories. Each issue report within the dataset is characterized by a title and a body, and is given one of the following labels, indicating the type of issue: *bug*, *feature*, *question*, or *documentation*.

To address the research questions of this study, we constructed multiple datasets with different characteristics and purposes, all derived from the NLBSE'23 dataset. All datasets employed in this study, including the expanded manually verified training set and the specialized long issues dataset described below, are made publicly available in the replication package at [18] to support reproducibility and future research.

4.1.1 Primary Dataset. For the main analysis, we started from a manually-verified subset of the NLBSE'23 dataset, published in prior research by Colavito et al. [15]. Using stratified sampling, Colavito et al. obtained this subset by extracting 400 issue reports – 200 as a training set and 200 as a test set – from the NLBSE'23 dataset. For each of the sampled issue reports, Colavito et al. manually re-evaluated the labels to address the presence of noisy or incorrect annotations initially submitted by GitHub users. This verification process highlighted the existence of several inconsistencies in the original labels.

To expand the manually verified dataset produced by Colavito et al. and to strengthen the reliability of the results, we applied stratified sampling to extract an additional subset of 400 issue reports from the NLBSE'23 dataset, disjoint from the subset extracted by Colavito et al.

Two of the authors of this paper manually verified these additional reports, following the same procedure defined by Colavito et al. in [15] with explicit annotation criteria.

The annotation process followed hierarchical decision rules based on primary user intent: Bug reports contained error descriptions or unexpected behavior without explicit help-seeking language. Questions were identified by interrogative phrases, help-seeking expressions (e.g., "*how to*", "*need help*"), or uncertainty statements, even when technical details or error logs were present. Feature requests contained explicit proposals for new functionality using terms like "*add*", "*implement*", or "*support*". Documentation issues involved requests for clarification or updates to project documentation. Edge cases where technical problems were presented alongside help-seeking language were classified as Questions, prioritizing user intent over technical content. Issues were labeled as discarded when they contained fewer than 10 meaningful words or mixed multiple intents without a clear priority.

After the independent annotation phase, we observed an almost perfect agreement between the annotators, with a Cohen's Kappa score [13] of 0.93. Disagreements occurred in 22 cases out of the 400 sampled issue reports (5.5% of cases) and were addressed by reaching a consensus through discussion. As done also in [15], issue reports for which it was not possible to reach a consensus, or for which both annotators agreed that there was not enough information to properly classify them into one of the considered classes, were assigned a *discarded* label.

As a result, the final primary dataset employed in our study consists of 600 manually verified issue reports for training (i.e., the original training set of 200 issue reports from [15], plus the 400 additional reports annotated by us) and 200 issue reports for testing (i.e., the original test set from [15]). By using the original test set, we aim at ensuring comparability of our results with those of other studies that employed the same dataset. The distributions of labels in the primary dataset (before and after manual re-annotations) are reported in Table 1.

To better characterize the issue report test set, we conducted an analysis of documentation features across different report types. In particular, two authors of this paper manually analyzed each issue report to identify whether it includes specific documentation elements, namely *detailed descriptions*, *logs*, and *stack traces*. More in detail, we consider an issue report to not contain a detailed description if it lacks information about the environment, steps to reproduce, expected and actual results, or any meaningful additional notes. After the annotation process, we observed a substantial agreement between the annotators, with an overall Kappa score of 0.85. Disagreements were solved through discussion. The results of this analysis are visualized in Figure 11.

4.1.2 Long Issues Dataset. The primary dataset is skewed toward shorter issue reports, with the majority containing fewer than 1000 tokens. This distribution reflects common patterns in open-source projects, where many issues are brief and lack detailed context. However, as discussed in Section 3.1, these shorter issues may not be representative of the more detailed and verbose issue reports often encountered in industrial and large-scale software projects.

Table 1. Distribution of labels in the manually-verified dataset

	Training Set								Test Set			
	Initial subset (re-labeled by Colavito et al. [15])				Additional subset (re-labeled by us)				Initial subset (re-labeled by Colavito et al. [15])			
	GitHub Labeling		Manual Re-labeling		GitHub Labeling		Manual Re-labeling		GitHub Labeling		Manual Re-labeling	
Bug	50	25%	47	24%	100	25%	113	28%	50	25%	53	27%
Documentation	50	25%	33	17%	100	25%	74	19%	50	25%	32	16%
Feature	50	25%	60	30%	100	25%	106	27%	50	25%	55	28%
Question	50	25%	44	22%	100	25%	93	23%	50	25%	47	24%
Discarded	–	–	16	8%	–	–	14	4%	–	–	13	7%
Total	200 issue reports				400 issue reports				200 issue reports			

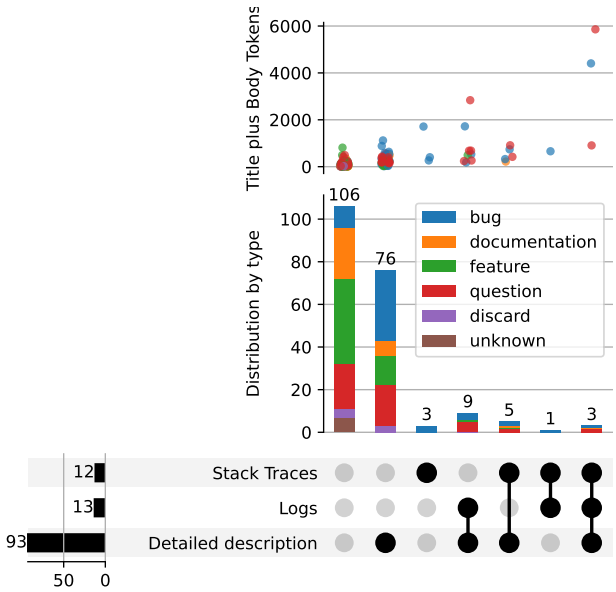

Fig. 11. Distribution of documentation features across different issue report types in our dataset.

To better capture this complexity and evaluate the robustness of our approach on longer inputs, we created a specialized dataset focusing on issues with substantial textual content. We extracted a stratified sample of 60 issues from the NLBSE'23 dataset [26], ensuring representation across both classification labels and issue length categories.

The stratification was performed along two dimensions:

- **Classification labels**: *bug*, *documentation*, *feature*, and *question*
- **Length categories** based on the total token count of the issue title and body:
  - Class 1: 4,001 to 10,000 tokens

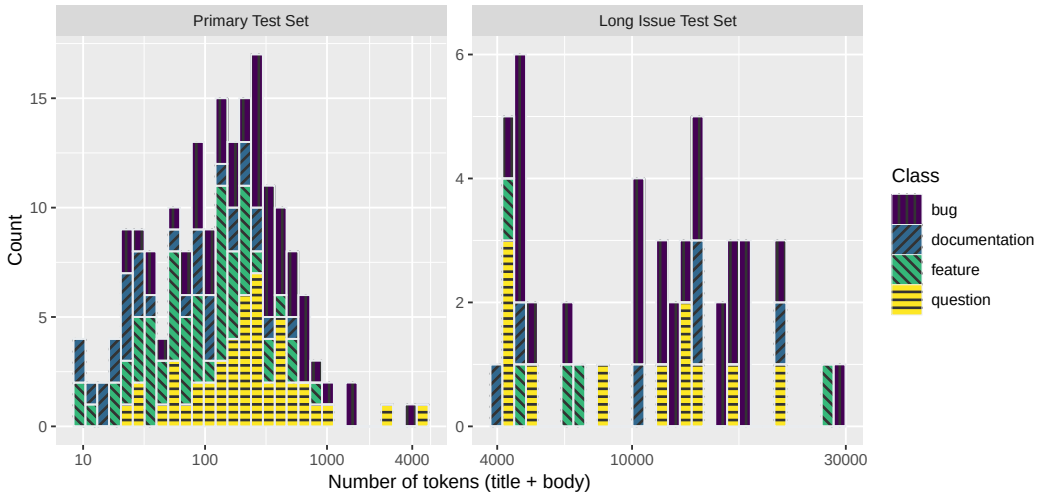

Fig. 12. Distribution of token lengths (title + body) in the primary and long issue test sets. The x-axis is log-scaled to account for the strong skewness in token counts.

- Class 2: 10,001 to 15,000 tokens
- Class 3: 15,001 tokens and above

For each combination of length class and classification label, we extracted five issue reports, resulting in a total of 20 issue reports per length class and 60 issue reports overall (5 reports × 4 labels × 3 length classes = 60 reports).

The labels assigned to the sampled issue reports were verified manually following the same rigorous procedure used for the primary dataset. Specifically, two authors of this paper independently annotated each of the 60 sampled issue reports. After the independent annotation phase, we calculated the inter-annotator agreement using Cohen's Kappa score [13], observing a high level of agreement with a Kappa score of 0.95. Issues for which consensus could not be reached were assigned a *discarded* label, resulting in 12 discarded issues from the original sample.

Figure 12 shows the distribution of token lengths in both the primary and long issue test sets. Due to the strong skewness in token counts, we adopted a log-scaled x-axis to improve readability. As the figure shows, the primary dataset is concentrated in the 100–1000 token range, with *bug* and *question* issues being generally associated with a higher token count than *feature* and *documentation* issues. The long issue dataset, on the other hand, spans from 4K to 30K tokens, capturing a broader spectrum of real-world complexity.

Qualitative Audit of Long Issue Reports. To assess whether over-length cases represent meaningful reports rather than noisy outliers, two authors audited the 48 retained long issues against the same documentation features used for the primary set (detailed descriptions, logs, stack traces), as presented in Section 4.1.1. During the audit, we observed substantial agreement ($\kappa = 0.87$), resolving disagreements through discussion.

The results of this quality audit are reported in Table 2. Only three issue reports[1] lacked all documentation features, and corresponded to cases where a large amount of boilerplate template text content was present. In contrast, the remaining reports were consistently rich in technical

[1]Specifically, IDs: #1243950176, #1154580678, and #1341487196.

Table 2. Presence of Documentation Features (Detailed Descriptions, Logs, Stack Traces) in Long Issues (n=48)

Feature Combination	Count	Percentage
No features	3	6%
At least one feature	45	94%
One feature	16	33%
Two features	20	42%
All three features	9	19%

detail: 94% included at least one meaningful technical artifact (i.e., detailed description, logs, or stack traces), 42% contained two distinct artifact types, and 19% incorporated all three. These findings confirm that lengthy reports are typically rich in actionable content rather than just being verbose.

To illustrate the nature of these long reports, Example 1 shows a representative case from the audit. In the example, excessively long code fragments, logs, and stack traces have been replaced with short red annotations between square brackets to indicate omitted content while preserving the overall structure of the issue report.

Example Issue Report 1: MineRL environment not loading, possible error with initializing GLFW (#1315527558)

So I have successfully installed the `minerl` package into my anaconda environment. I am running Python 3.9.12. I am working on a remote server where I will then use X11 forwarding to display anything the `minerl` package needs to display. The following code runs successfully:

```
[Snippet of code to load the minerl package omitted - 96 tokens]
```

All good, right? However, when I then run `obs = env.reset()`, I am first greeted with this warning:

```
RuntimeWarning: 'minerl.utils.process_watcher' [... rest of the warning message omitted - 76 tokens]
```

Some research as well as discussion on the minerl discord server has told me that this warning is normal, and even pops up on successful ‘minerl‘ runs. Ok. However, after this warning, what is not normal is that I get the following error, and I cannot get ‘minerl‘ to work.

```
INFO:process_watcher:Process watcher daemonizer launched successfully.
DEBUG:minerl.env.malmo.instance:SLF4J: Defaulting to no-operation (NOP) logger implementation
[... rest of the log and stack trace omitted - approx. 4300 tokens]
```

I want to say that the issue root of the issue is here:

```
java.lang.IllegalStateException: Failed to initialize GLFW
[... rest of the stack trace details omitted - 100 tokens]
```

The issue is similar to the one mentioned [here](https://github.com/minerllabs/minerl/issues/646), but it is not the same. I have looked into "GLFW" errors and come across [this](https://github.com/LWJGL/lwjgl3/issues/499), which suggests installing a new version of LWJGL ([Lightweight Java Game Library](https://www.lwjgl.org/)). I don’t know how to do this though or where I’d install it. But this could very well be a red herring as well. This is the info about my java version: openjdk version "1.8.0_332"; OpenJDK Runtime Environment (Zulu 8.62.0.19-CA-linux64) (build 1.8.0_332-b09); OpenJDK 64-Bit Server VM (Zulu 8.62.0.19-CA-linux64) (build 25.332-b09, mixed mode).

Has anyone encountered this issue? If so, did you find a work-around? Thanks in advance.

This issue spans over 5,100 tokens and includes multiple technical artifacts: a detailed problem description, environment specifications, and an extensive error log culminating in a Java stack trace. Beyond the presence of technical information, the example also illustrates a recurring rhetorical pattern: the communicative intent (i.e., whether to request guidance, propose a new feature, or confirm the presence of a bug) can be located toward the end of the report, after extensive diagnostic detail. In other words, the decisive signal for classification may appear only after environment descriptions, reproduction steps, and long error logs. This reinforces that over-length cases are not merely verbose but combine rich technical content with intent cues that may appear throughout the report, making head-only truncation insuff0icient for capturing all relevant signals in downstream tasks.

4.2 LLMs Selection

In order to assess the performance of the proposed methods across a range of model architectures, capabilities, and deployment scenarios, we used three distinct LLMs: GPT-4o, GPT-3.5-turbo (henceforth simply referred to as GPT-3.5 for brevity), and Qwen-2.5 (32B parameters) [35].

The OpenAI API was used to access GPT-4o and GPT-3.5. We chose to use GPT-4o because, as OpenAI also reported, we expected the newer version to perform better. This choice allowed us to leverage the latest advancements in language model technology and ensured that our methods benefit from enhanced effectiveness and robustness. Conversely, we selected GPT-3.5 to benchmark the performance of our methods against the approach proposed by Colavito et al. [16]. However, it is noteworthy that OpenAI has announced the deprecation of GPT-3.5. Despite its impending deprecation, GPT-3.5 remains a relevant benchmark due to its widespread use and established performance metrics in prior research.

The inclusion of Qwen-2.5 represents a strategic decision to evaluate our methods with an open-weight alternative that offers different architectural characteristics. Indeed, Qwen-2.5 is a decoder-only transformer model with 32 billion parameters, and offers a significant capacity for intricate reasoning tasks while still being deployable in local settings. We deployed the model locally using LM Studio (i.e., a comprehensive platform for LLM experimentation that provides a unified OpenAI-compatible REST API for model deployment and inference across various architectures), with the aim of simulating enterprise or sensitive data environments, where privacy concerns and computational autonomy are critical requirements. We conducted additional validation experiments with more recent open-source models, including Qwen3 variants (14B, 32B, 235B), Gemma3 (4B, 27B), DeepSeek R3.1 (671B), Llama 4 (401B), OpenAI GPT-OSS (20B), Ernie 4.5 (21B), and Mistral 2501 (24B), following the same classification performance procedure described in Section 4.4.1. We excluded code-specialized variants such as Qwen3-Coder as they are optimized for code generation rather than natural language understanding tasks. The results, reported in the replication package, confirm that Qwen-2.5 (32B) maintains optimal performance-cost-stability trade-offs across all three methods compared to these alternatives.

4.3 Baselines

As baselines, we selected three state-of-the-art approaches that have been recently presented in the literature. In particular, we consider the following baselines: (1) the RoBERTa-based approach which is also used as a baseline in the NLBSE'23 Issue Report Classification challenge [26], (2) the SetFit-based approach proposed in [15], and (3) the ChatGPT-based approaches proposed in [16]. Details of each baseline approach are discussed in Section 2.

4.4 Experimental Procedure and Metrics

This section outlines the procedures and metrics used to address the research questions.

4.4.1 RQ1: Method Effectiveness and Operational Characteristics. RQ1 investigates the effectiveness and operational characteristics of the proposed methods through six analyses: (1) classification performance evaluation and temperature sensitivity analysis, (2) baseline comparison, (3) intent extraction ablation study, (4) long issue handling evaluation, (5) token usage and cost assessment, and (6) perplexity-based analysis. The performance metrics and procedure for these analyses are explained in the following sections.

Classification Performance Evaluation and Temperature Sensitivity Analysis. To rigorously assess the classification performance of our three proposed methods, we conducted a comprehensive evaluation on a manually verified test set of 200 issue reports (see Section 4.1.1). Since the optimal value of the temperature parameter, which controls the degree of randomness in LLM token selection, is not known a priori for this task, we systematically explored its impact through a temperature sensitivity analysis. Specifically, each method was evaluated using three LLMs (GPT-3.5, Qwen-2.5, and GPT-4o as discussed in Section 4.2) across four representative temperature settings: 0 (minimal randomness), 0.3 and 0.7 (moderate stochasticity levels that may improve performance on tasks requiring diverse reasoning approaches), and 1.0 (unscaled probability distribution providing a natural baseline for stochastic sampling behavior).

For each method-model-temperature configuration, we performed five independent runs to ensure robust and reliable estimates. Note that, while setting the temperature to 0 significantly reduces randomness, recent research demonstrates that complete determinism cannot be guaranteed even at a temperature of 0 due to inherent implementation details in LLM inference [34]. All this means that multiple runs remain valuable, even with the temperature set to 0, to account for residual non-deterministic behavior and provide more robust performance estimates. For each repetition, we computed F1-scores for each class, as well as macro-averaged and weighted-averaged F1-scores, enabling a detailed comparison of classification effectiveness and the identification of the best-performing temperature for each method-model combination. This joint analysis provides both a comprehensive assessment of classification performance and empirical guidance for selecting optimal temperature settings in subsequent experiments.

Comparison with Baselines. A separate comparative evaluation assessed the proposed methods' performance – measured in the *Classification Performance Evaluation and Temperature Sensitivity Analysis* – against the three established baselines reported in Section 4.3. The RoBERTa-based, SetFit-based, and GPT-3.5-based approaches were implemented using their original configurations and trained on the expanded, manually verified training set of 600 issue reports (see Section 4.1.1). These baselines were then evaluated on the same test set using the same metrics, allowing for a direct performance comparison with the proposed methods. This analysis specifically aimed to demonstrate whether the proposed approaches could achieve superior classification effectiveness compared to existing state-of-the-art methods.

Intent Extraction Ablation Study. To isolate the contribution of the intent extraction component to the overall performance of Methods 1 and 2, we conducted an ablation study. For each method-LLM combination, two versions were evaluated on the same test set of 200 issue reports illustrated in Section 4.1.1: one with the complete pipeline, including intent extraction, and one without the intent extraction component. The ablation study measured classification performance under both conditions across all three LLMs, computing the same precision, recall, and F1-measure metrics.

Long Issue Handling Evaluation. To evaluate the effectiveness of the MapReduce-inspired approach for handling long issue reports, we conducted classification analyses using Method 1 and Method 3 on the long issues test set described in Section 4.1.2. This dataset was specifically constructed to include only issue reports with a minimum length of 4,001 tokens, ensuring that each

sample exceeds the default context window of 4,000 tokens and thus is eligible for the multi-step summarization strategy. Method 2 was not included in this evaluation since it functions as an ensemble of Method 1 instances and preserves the same MapReduce benefits without altering the underlying preprocessing strategy.

The experiments were divided into two conditions:

- **With the MapReduce Approach**: We applied the MapReduce-inspired pipeline described in Section 3.1 to classify the issues.
- **With Truncation**: We truncated the combined title and body of each issue to a maximum of 4,000 tokens before classification. Truncation represents a simplified approach that does not utilize the proposed approach for handling varying lengths, relying only on the initial portion of the description for classification.

The classification results were evaluated using the same metrics described in **??**. In addition, we evaluated classification results of the baselines, the RoBERTa model [26] and SETFIT [15], using the same dataset and metrics, to verify their effectiveness on long text descriptions.

To further investigate whether longer issue descriptions can be effectively processed by LLMs with extended context windows, we performed an additional analysis using GPT-4o with direct ingestion of the full issue text, leveraging its full context capacity of 128k tokens. This examination was conducted leveraging only GPT-4o, as the context size of the other considered LLMs (16k tokens for GPT-3.5, 32k tokens for Qwen-2.5) is not sufficient to fit all the issues in our longer issues test set.

Token Usage and Cost Analysis. To investigate the practical feasibility and sustainability of the proposed methods and to provide practical guidance to practitioners interested in LLM-based issue classification systems at scale, we analyzed both token consumption patterns and associated classification costs.

For token usage analysis, we tracked the total number of input and output tokens required by each LLM/method combination to process every issue in our test sets (see Section 4.1.1). This granular token accounting allows us to understand the computational overhead of different approaches and identify potential optimization opportunities. The analysis was conducted separately for the longer issues test set (see Section 4.1.2), comparing both the truncation and MapReduce approaches to investigate their respective resource consumption profiles.

To translate token usage into practical cost considerations, we adopted a token-based accounting approach grounded in the pricing models of commercial LLM-as-a-service offerings. Token counts were multiplied by reference per-token costs to derive cost estimates that reflect real-world deployment scenarios. The reference costs for GPT-based models were obtained from Microsoft Azure's publicly available pricing for their language model service offerings [31]. For Qwen-2.5, which was deployed locally, we simulated a realistic cloud-based deployment scenario by applying the pricing model offered by Cloudflare Workers AI [12]. This approach ensures fair and consistent cost comparisons across all models. The reference rates (in USD) we considered are:

- GPT-3.5: $ 1.50 per 1M input tokens, $ 2.00 per 1M output tokens;
- Qwen-2.5: $ 0.66 per 1M input tokens, $ 1.00 per 1M output tokens;
- GPT-4o: $ 2.50 per 1M input tokens, $ 10.00 per 1M output tokens.

Perplexity Analysis Experiment. To gain a more in-depth insight into LLM-based classification performance, we conducted additional experiments using our test set (see Section 4.1.1) to investigate whether a model's internal perplexity metric [23] – which measures model uncertainty in predictions – differs between correct and incorrect predictions. This analysis could provide critical insights into the reliability of this metric as a measure of decision certainty in the issue report classification

domain, with implications for both model interpretability and evaluation. More formally, the perplexity metric we considered is defined as follows:

$$Perplexity = \exp\left(-\frac{1}{n}\sum_{i=1}^{n}\log p_i\right),$$

where:

- n is the number of tokens in the generated sequence
- $\log p_i$ is the log probability of the i-th token as assigned by the language model

Perplexity is a standard metric in natural language processing that measures how well a probability model predicts a sample [23]. Lower perplexity values indicate higher model confidence, as the model assigns higher probabilities to the generated tokens.

For this metric, we computed three variants:

- *Class*: Perplexity is computed limited to the final classification step, using only the log probabilities from the classification model's final output tokens.
- *Summary or Intent*: Perplexity is computed concerning the summarization step (or intent extraction step, depending on the method used), using only the log probabilities from the summary generation tokens.
- *Total*: Perplexity is computed by combining the log probabilities from both the summarization (or intent) and classification steps, taking the mean of all log probabilities across both tasks to provide an overall confidence and perplexity measure for the entire pipeline.

We note that these experiments are limited to GPT-3.5 and GPT-4o, which are the only models for which it was possible, through the OpenAI APIs, to access token-level log probabilities. While open-source alternatives that provide OpenAI-compatible APIs exist (such as vLLM, Text Generation Inference, or Ollama), most current implementations do not expose token-level log probabilities in their API responses. Although ongoing development work is underway to incorporate this functionality into open-source serving frameworks, at the time of this study, reliable access to log probabilities was only available through OpenAI's commercial API endpoints.

To assess whether perplexity values differ between correct and incorrect classifications, for each LLM and method, we stratified the test set into two groups based on prediction correctness. We then applied Wilcoxon Rank-Sum Tests to evaluate the alternative hypothesis that perplexity is significantly lower for correctly classified instances. To account for multiple comparisons, we employed the Holm-Bonferroni correction with an initial significance level of $\alpha = 0.05$.

Building on insights from the perplexity analysis, we conducted an additional experiment to evaluate whether Method 2's ensemble approach could be improved by replacing simple majority voting with perplexity-based alternatives (i.e., selecting the response with the smallest perplexity). This experiment compared the alternative voting mechanisms for Method 2 across GPT-3.5 and GPT-4o, measuring F1-scores for each class and computing weighted-averaged performance to assess whether confidence-weighted decisions could improve ensemble performance.

4.4.2 RQ2: Classification Consistency. RQ2 aims to assess the consistency of classification results across multiple runs to ensure the proposed methods deliver the stable and predictable outcomes required for practical deployment in software engineering environments.

To quantify classification consistency, we employed two complementary metrics commonly used in Software Engineering studies. The raw percentage of agreement was calculated as the percentage of issue reports for which all model repetitions agreed on the same label [46]. This metric provides a straightforward overview of classification consistency across different runs but does not account for the possibility of agreements occurring by chance alone. To address this limitation, we computed

the Fleiss' Kappa inter-rater agreement index [20], a statistical measure that evaluates the level of agreement between multiple raters while accounting for chance agreements. The Fleiss' Kappa score ranges between 0 and 1, with a value of 1 indicating perfect agreement and a value of 0 representing agreement occurring purely by random chance [29].

The core consistency evaluation analyzed the variability in classification outcomes across the five repetitions performed in Section 4.4.1. Each repetition was treated as a distinct annotator, allowing us to measure inter-rater agreement between different classification runs using the established metrics. This analysis was conducted for each combination of the three proposed methods and three LLMs, in order to provide insights in terms of stability. The analysis examined whether certain method-model combinations exhibited superior consistency properties, which could influence their suitability for production deployment despite potentially comparable performance.

To complement the analysis of temperature effects on classification performance, we investigated how temperature variations affect the consistency of predictions across multiple runs. The same temperature settings used in section 4.4.1 were employed: 0.0, 0.3, 0.7, and 1.0. For each temperature-method-model combination, we computed both raw percentage agreement and Fleiss' Kappa scores across the five repetitions. This investigation aimed to determine whether lower temperature settings, which reduce randomness in token selection, correspondingly improve prediction consistency and whether this relationship holds uniformly across all methods and models. The analysis provided insights into the trade-off between classification performance and consistency at different temperature levels, informing optimal parameter selection for practical applications where both performance and reliability are critical considerations.

4.4.3 RQ3: Error Patterns and Classification Dynamics. The goal of RQ3 is to uncover misclassification patterns, highlight recurring challenges, and identify specific weaknesses in the proposed methods, which could guide future improvements in issue report classification systems.

To this end, we adopted a multi-level, progressive-refinement approach, focusing on the issues that were consistently misclassified by the proposed methods in the evaluation described in Section 4.4.1, i.e., for which the classification was incorrect in all repetitions. First, we analyzed confusion matrices to identify which class pairs were most frequently confused by examining how often issues of each actual class were incorrectly predicted as other classes. This quantitative analysis revealed the most problematic class distinctions and provided an overview of systematic classification errors.

To gain deeper insights into the nature of misclassifications, we conducted a fine-grained manual inspection of the consistently incorrect predictions. This analysis was guided by a structured annotation protocol designed to ensure systematic and reliable identification of the underlying causes behind each misclassification. Initially, two authors jointly annotated a randomly sampled subset of 20 consistently misclassified issues to develop a taxonomy of error causes and establish a shared understanding of classification criteria. This initial coding process focused on identifying specific characteristics that could explain classification failures. After establishing these criteria through joint annotation of the initial subset, the authors independently annotated the remaining misclassified issues. To assess the stability and consistency of the independent annotations, we computed the inter-rater agreement score using Cohen's Kappa. Disagreements were resolved through discussion until complete consensus was achieved.

Subsequently, we performed a combined analysis, linking class-level misclassification patterns with the fine-grained taxonomy. This combined analysis allowed us to investigate whether different types of misclassification errors were more linked to specific class confusions.

Finally, we conducted a cross-method comparative analysis examining error patterns across the three proposed methods and three LLMs. This analysis investigated several dimensions: (1) how the

same misclassified issues were handled differently by each method-LLM combination; (2) whether Method 2's ensemble approach corrected or consolidated errors from Method 1; (3) how Method 3's few-shot learning affected error redistribution across different models; and (4) the relationship between issue characteristics (token length, presence of technical artifacts) and classification robustness. The analysis examined both individual method failures and consensus failures where all LLMs misclassified the same issue, aiming to identify systematic biases, method-specific failure modes, and potential complementary strengths across different approaches. Additionally, for GPT-3.5 and GPT-4o, we analyzed perplexity patterns to investigate uncertainty calibration differences between correct and incorrect predictions across the different methods.

5 Analysis of the Results

In this section, we present the results for each research question.

5.1 RQ1: Method Effectiveness and Operational Characteristics

In the following, we report the results for the evaluations illustrated in Section 4.4.1. For the sake of conciseness, we only report F1 scores, while the precision and recall scores are available in the replication package.

5.1.1 Classification Performance Evaluation and Temperature Sensitivity Analysis. The results of the temperature (T) sensitivity analysis are reported in Table 3, showing weighted-averaged F1-scores across four temperature settings (0.0, 0.3, 0.7, 1.0) for all method-LLM combinations.

Table 3. Weighted-averaged F1-scores achieved by the proposed methods across different temperature (T) settings.

	Method 1			Method 2			Method 3		
T	**GPT-3.5**	**Qwen-2.5**	**GPT-4o**	**GPT-3.5**	**Qwen-2.5**	**GPT-4o**	**GPT-3.5**	**Qwen-2.5**	**GPT-4o**
0.0	0.80	0.89	0.92	0.86	0.89	0.92	0.86	0.92	0.95
0.3	0.82	0.89	0.91	0.81	0.90	0.92	0.86	0.71	0.92
0.7	0.80	0.88	0.91	0.81	0.89	0.92	0.86	0.71	0.93
1.0	0.80	0.88	0.91	0.81	0.91	0.92	0.85	0.71	0.93

GPT-4o demonstrates remarkable stability across all temperature settings and methods, with performance remaining essentially unchanged (0.91-0.95 range). GPT-3.5 shows moderate sensitivity, particularly with Method 2, where performance drops from 0.86 (T=0.0) to 0.81 at higher temperatures.

Most notably, Qwen-2.5 exhibits a unique architecture-method interaction. While remaining stable with Methods 1 and 2 across all temperatures, it suffers performance degradation with Method 3 when the temperature exceeds 0.0 (F1-score drops from 0.92 to 0.71). This behavior suggests a specific incompatibility between Qwen-2.5's architecture and few-shot learning with explanations under stochastic sampling conditions. These findings indicate that, although temperature sensitivity varies for some method-LLM combinations, the classification task appears to benefit primarily from consistent reasoning rather than creative exploration. Given that T=0.0 provides better performance in most method-model combinations, we adopt T=0.0 for all subsequent experiments.

The detailed classification performance results for the three proposed methods across different LLMs, with temperature see to 0, are reported in Table 4 (rows 4-12). Method 3 (Explained Few-Shot

Table 4. Classification results across four issue classes, reported in terms of precision, recall, and F1-score. Method names are abbreviated as M1, M2, and M3. The table also reports the weighted-average F1 across classes, and its variation (Δ) against the best-performing baseline (SetFit). Positive Δ values indicate performance gains. Within each metric column, the highest score is highlighted in bold.

	Bug			Documentation			Feature			Question			Weighted	Weighted Average F1
Model	**Prec.**	**Recall**	**F1**	**Prec.**	**Recall**	**F1**	**Prec.**	**Recall**	**F1**	**Prec.**	**Recall**	**F1**	**Average F1**	**Δ vs. SetFit**
RoBERTa [26]	0.92	0.83	0.87	0.63	0.81	0.71	0.84	0.76	0.80	0.81	0.83	0.82	0.81	-0.06
GPT 3.5-based [16]	0.74	0.96	0.83	0.93	0.66	0.77	0.87	0.81	0.84	0.81	0.78	0.79	0.82	-0.05
SetFit [15]	0.94	0.85	0.89	0.86	0.78	0.82	0.83	0.89	0.86	0.86	0.94	0.90	0.87	–
M1 GPT-3.5	0.76	**0.97**	0.85	0.92	0.72	0.81	0.77	0.94	0.84	0.98	0.54	0.70	0.80	-0.07
M2 GPT-3.5	0.86	0.96	0.91	0.88	0.77	0.82	0.84	0.87	0.86	0.88	0.81	0.84	0.86	-0.01
M3 GPT-3.5	0.77	**0.97**	0.86	0.98	0.81	0.88	0.88	0.93	0.91	**0.93**	0.70	0.80	0.86	-0.01
M1 Qwen-2.5	0.88	0.96	0.92	0.86	0.86	0.86	0.89	0.87	0.88	**0.93**	0.85	0.89	0.89	+0.02
M2 Qwen-2.5	0.88	0.96	0.92	0.86	0.86	0.86	0.89	0.87	0.88	**0.93**	0.85	0.89	0.89	+0.02
M3 Qwen-2.5	**0.96**	0.87	0.91	0.93	0.93	0.93	0.98	0.91	0.94	0.82	0.98	0.89	0.92	+0.05
M1 GPT-4o	0.93	0.88	0.90	0.96	0.90	0.93	**0.96**	0.91	0.94	0.83	**0.97**	0.90	0.92	+0.05
M2 GPT-4o	0.93	0.89	0.91	0.96	0.90	0.93	**0.96**	0.91	0.93	0.83	**0.97**	0.90	0.92	+0.05
M3 GPT-4o	**0.96**	0.92	**0.94**	**0.97**	**0.99**	**0.98**	**0.96**	**0.95**	**0.95**	0.91	0.96	**0.94**	**0.95**	**+0.08**

Learning) achieves the best overall performance, particularly with GPT-4o (0.95 weighted F1-score), representing a 3% improvement over the other methods. Method 2 (Ensemble Classification) consistently outperforms Method 1 across all LLMs, with the most significant gains observed with GPT-3.5 (6% improvement: 0.86 vs 0.80 weighted F1-score). In addition to overall F1 performance, the per-class precision and recall values reveal consistent trends across models and LLMs. All methods achieve their strongest and most stable performance on Bug and Feature reports, with several configurations (i.e., particularly Method 3 with GPT-4o) reaching precision and recall values above 0.95. Performance on Documentation and Question is more variable across models and LLMs. For Documentation, Method 3 with GPT-4o achieves its strongest classification performance, reaching an F1-score of 0.98, the highest among all configurations. In contrast, GPT-3.5 exhibits a consistent tendency toward higher precision than recall for this class (e.g., 0.98 precision vs. 0.81 recall with Method 3), indicating that it predicts Documentation more conservatively. For the Question class, GPT-4o generally achieves higher recall than precision (e.g., 0.97 recall vs. 0.83 precision with Method 1, and 0.96 vs. 0.91 with Method 3), suggesting a greater ability to capture this category correctly. GPT-3.5 shows the opposite trend, with recall dropping as low as 0.54 (Method 1) despite strong precision values, pointing to systematic under-detection. Qwen remains comparatively stable across methods, with precision–recall pairs typically around 0.93–0.85 for Question and 0.86–0.86 for Documentation, though its overall F1-scores remain below those of GPT-4o, reflecting a modest but consistent performance gap.

The results reveal interesting patterns across model architectures. GPT-4o demonstrates the highest performance ceiling (0.92-0.95 weighted F1-scores). Despite having a different decoder-only architecture and fewer parameters, Qwen-2.5 outperforms GPT-3.5 across all methods (0.89-0.92 vs. 0.80-0.86). Furthermore, GPT-3.5's performance suggests it is more sensitive to method selection and that its effectiveness depends more on the specific prompt engineering technique used. It is important to note, however, that architectural differences alone do not fully explain this performance disparity between Qwen-2.5 and GPT-3.5. Multiple confounding variables may contribute to these findings, including differences in training corpora composition, knowledge cutoff dates, training objectives, fine-tuning approaches, and various architectural innovations. Taking these factors into consideration, we interpret our results as suggesting that more recent

models, when thoughtfully designed, can achieve comparable or superior performance to older, larger models in issue classification tasks, even with substantially fewer parameters.

5.1.2 Comparison with Baselines. Table 4 reports classification performance for both the baseline models (RoBERTa, SetFit, and the LLM-based approach from [16], rows 1–3) and the proposed methods (rows 4–12). When comparing the proposed methods against these baselines, results show that the proposed approaches can achieve superior performance, with varying degrees of improvement depending on the baseline and model combination.

All proposed methods with GPT-4o significantly outperform the best baseline (SetFit: 0.87 weighted F1-score), achieving improvements ranging from 5% (Methods 1 and 2: 0.92) to 8% (Method 3: 0.95). Similarly, Qwen-2.5 outperforms all baselines across all methods, with weighted F1-scores ranging from 0.89 to 0.92, representing improvements of 2% to 5% over the best baseline. When using GPT-3.5, performance varies by method. Method 1 (0.80 weighted F1-score) underperforms compared to SetFit (0.87), while Methods 2 and 3 (both 0.86) approach SetFit's performance but remain slightly below. Notably, SetFit excels specifically in question classification (0.90 F1-score), while our GPT-3.5 methods show more balanced performance across other classes.

Compared to the existing LLM-based approach by Colavito et al. (0.82 weighted F1-score with GPT-3.5), the Methods 2 and 3 achieve meaningful improvements (both 0.86), representing a 4% gain. All this demonstrates that both ensemble approaches and few-shot learning with explanations provide tangible benefits over basic LLM prompting strategies, and establishes that the proposed methods achieve state-of-the-art performance, with the magnitude of improvement being architecture-dependent and method-specific.

5.1.3 Impact of Intent Extraction on LLM-based Issue Report Classification. The results of the ablation study described in Section 4.4.1 are reported in Table 5. For the sake of conciseness, we only report weighted-averaged precision, recall, and F1 scores. Class-level metrics are available in the replication package.

Table 5. Weighted-average precision, recall, and F1 for Method 1 (M1) and Method 2 (M2) with and without intent extraction, across all LLMs.

	With Intent Extraction			**Without Intent Extraction**		
Model	**Precision**	**Recall**	**F1**	**Precision**	**Recall**	**F1**
M1 GPT-3.5	0.84	0.81	0.80	0.80	0.75	0.73
M2 GPT-3.5	0.87	0.86	0.86	0.78	0.73	0.71
M1 Qwen-2.5	0.89	0.89	0.89	0.85	0.85	0.85
M2 Qwen-2.5	0.89	0.89	0.89	0.87	0.87	0.87
M1 GPT-4o	0.92	0.92	0.92	0.90	0.90	0.90
M2 GPT-4o	0.92	0.92	0.92	0.89	0.89	0.89

GPT-3.5 shows the most significant improvements by using intent extraction. Method 1 improves from 0.80→0.84 in precision, 0.75→0.81 in recall, and 0.73→0.80 in F1, while Method 2 demonstrates even larger gains (precision: 0.78→0.87; recall: 0.75→0.86; F1: 0.71→0.86). These substantial improvements indicate that GPT-3.5 particularly benefits from the structured approach of explicitly extracting user intent before performing classification.

Qwen-2.5 and GPT-4o0 exhibit smaller but meaningful improvements. For Qwen-2.5, both methods show consistent 2%-4% increases across precision, recall, and F1. For GPT-4o, improvements

Table 6. Weighted average F1 scores for issue classification methods across different test sets and input processing approaches. The highest F1 scores in each column are highlighted in bold.

Model	Primary Test Set	Longer Issues Test Set	
		MapReduce Approach	Truncation Approach
RoBERTa [26]	0.81	–	0.75
SetFit [15]	0.87	–	0.79
Method 1 GPT-3.5	0.80	0.82	0.71
Method 1 Qwen-2.5	0.89	0.81	0.78
Method 1 GPT-4o	0.92	0.89	0.87
Method 3 GPT-3.5	0.86	0.84	0.66
Method 3 Qwen-2.5	0.92	0.90	0.79
Method 3 GPT-4o	**0.95**	**0.94**	**0.85**

range from 1% to 3%. These results suggest that different architectural designs respond differently to structured intent information, with GPT-4o's advanced capabilities providing some compensation for missing intent extraction, while Qwen-2.5's decoder-only architecture shows consistent benefits from the structured approach.

Method 2 generally shows greater improvements from intent extraction compared to Method 1, with the effect being most pronounced for GPT-3.5, where Method 2's improvement in F1 (15%) substantially exceeds Method 1's (7%). However, this pattern is not consistent across all models – Qwen-2.5 shows the opposite trend, with Method 1 benefiting more from intent extraction (F1: +4%) than Method 2 (F1: +2%). This suggests that the interaction between ensemble voting and intent extraction is model-dependent rather than universally amplifying.

5.1.4 Handling Long Issue Descriptions. The results of the evaluation on longer issue reports are reported in Table 6. The table shows the weighted average F1 scores for the three LLM-based methods across the primary test set (for consistency check and as baseline for comparison), the longer issues test set with MapReduce-like processing, and the longer issues test set with simple truncation, alongside baseline comparisons with RoBERTa and SetFit. To complement the tabular results and improve interpretability, we provide a comparative visualization in Figure 13 that reports the differences in weighted average F1 scores between each proposed method–LLM combination and the RoBERTa and SetFit baselines across all evaluation settings.

The results show that, across all methods and LLMs, the MapReduce approach provides substantial benefits when handling longer issue reports, outperforming simple truncation with improvements ranging from 2-11 percentage points with Method 1 and 9-18 percentage points with Method 3. This validates our hypothesis that preserving information for longer issues through the MapReduce strategy is crucial for preserving classification effectiveness.

Comparing these results with those achieved on the primary test set, we observe that the proposed methods generally maintain comparable or even improved performance on longer issues, while simple truncation leads to substantial degradation. In contrast, the baselines show notable performance degradation on longer issues, with RoBERTa dropping from a weighted average F1 score of 0.81 on the primary test set to 0.75 on the longer issues test set, and SetFit exhibiting a similar decline from 0.87 to 0.79, highlighting the inherent limitations of these approaches in dealing with extended input sequences that exceed their fixed context windows.

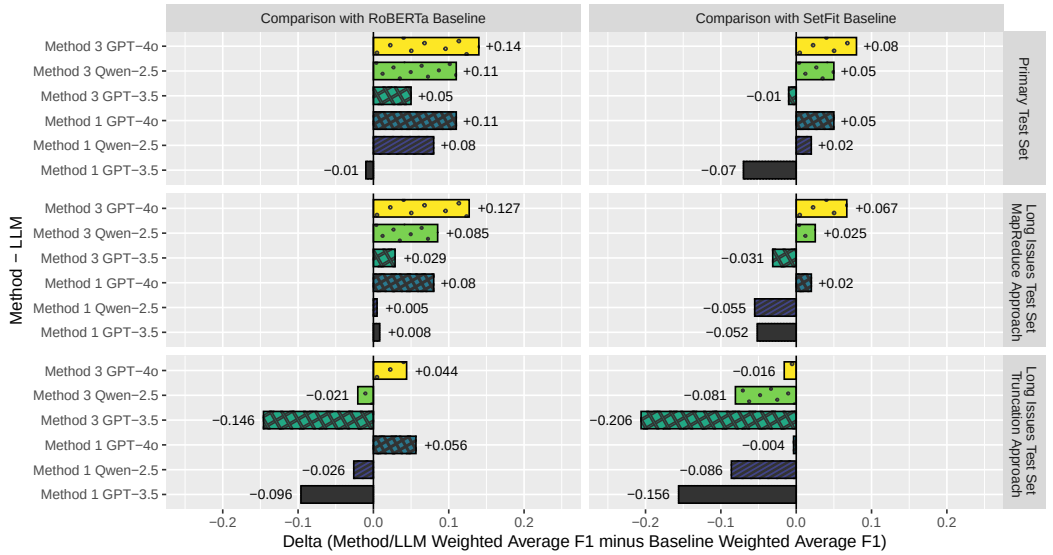

Fig. 13. Differences in weighted average F1 scores between each proposed method–LLM combination and the RoBERTa and SetFit baselines, across three evaluation settings: Primary Test Set, Longer Issues Test Set (MapReduce Approach), and Longer Issues Test Set (Truncation Approach). Positive values indicate performance improvements over the baseline.

Table 7. Weighted average F1 scores achieved by GPT-4o for longer issue classification using three input processing strategies: MapReduce-style decomposition, truncation, and direct ingestion of full issue descriptions.

Method	MapReduce	Truncation	Direct Ingestion
Method 1	0.89	0.87	0.80
Method 3	0.94	0.85	0.89

The results of the additional experiment aimed at assessing the effectiveness of directly ingesting full-length issue descriptions are reported in Table 7, and confirm that the MapReduce approach yields better performance. Indeed, classification F1 scores are consistently lower when using the full-context input compared to MapReduce-style processing.

For Method 1, the weighted average F1 score drops from 0.89 with MapReduce to 0.80 with full input, which is also lower than the truncation result of 0.87. For Method 3, direct ingestion of the full issue descriptions yields an F1 score of 0.89, placing it between the truncation approach (0.85) and the more effective MapReduce strategy (0.94). This finding aligns with observations from the literature [2, 43], where LLM performance has been observed to degrade as input context length increases, and reinforces the importance of structured input decomposition for long-form content, even when using models capable of handling large contexts.

5.1.5 Token Usage and Cost Analysis. Table 8 reports the average cost (in USD) of classifying a single issue report across different LLMs and processing strategies. Across all models and test sets (see Section 4.1), Method 1 is consistently the least expensive, with average costs ranging from $0.001 (Qwen-2.5) to $0.005 (GPT-4o) on the primary test set. In contrast, Method 3 incurs slightly higher costs, with primary test set values ranging from $0.003 (Qwen-2.5) to $0.010 (GPT-4o). This

Table 8. Average cost (in USD) to classify a single issue report using different LLMs, methods, and processing strategies.

					Longer Issues Test Set					
		Primary Test Set			MapReduce Approach			Truncation Approach		
LLM	Method	Token Input	Token Output	Cost (USD)	Token Input	Token Output	Cost (USD)	Token Input	Token Output	Cost (USD)
GPT-3.5	Method 1	1,746	30	$0.003	11,493	204	$0.018	4,768	41	$0.007
	Method 2	8,730	150	$0.013	57,466	1,020	$0.088	23,840	206	$0.036
	Method 3	3,906	4	$0.006	12,163	54	$0.018	7,578	3	$0.011
Qwen-2.5	Method 1	1,785	34	$0.001	11,526	227	$0.008	4,768	47	$0.003
	Method 2	8,926	170	$0.006	57,629	1,135	$0.039	23,840	235	$0.016
	Method 3	3,962	5	$0.003	12,267	130	$0.008	7,578	7	$0.005
GPT-4o	Method 1	1,750	32	$0.005	11,530	222	$0.031	4,768	38	$0.012
	Method 2	8,749	160	$0.023	57,651	1,110	$0.155	23,840	190	$0.062
	Method 3	3,855	1	$0.010	11,901	1	$0.030	7,569	1	$0.019

increase is attributed to its use of few-shot prompting, which includes multiple examples in the input and thus inflates token usage. The most expensive strategy is Method 2, which involves an ensemble of five Method 1 executions. This design leads to significantly higher costs, with primary test set values reaching $0.023 (GPT-4o) and $0.013 (GPT-3.5). These trends hold across the longer issues test set as well, where Method 2 under the MapReduce approach reaches up to $0.155 per issue (GPT-4o), emphasizing the need to carefully weigh performance gains against economic sustainability.

Focusing on the longer issues test set, the results show that the MapReduce approach, while guaranteeing superior classification performance w.r.t. simple truncation (as discussed in Section 5.1.4), incurs in higher inference costs due to the increased number of processed tokens. Across all models and methods, inference costs with the MapReduce strategy are approximately 2 to 2.5 higher than simple truncation.

5.1.6 Analysis of Perplexity. The analysis investigated whether LLM internal perplexity metrics can predict classification performance and improve ensemble voting mechanisms. Results are reported in Table 9. Note that, for the sake of conciseness, in the table we refer to the *Summary/Intent* perplexity as "*summary perplexity*".

Class-level perplexity emerges as a reliable indicator of classification performance. Statistical tests reveal significant differences ($p < 0.001$) between correct and incorrect predictions for class perplexity across most method-LLM combinations, with consistently lower perplexity values associated with correct classifications. Effect sizes are particularly large for GPT-4o with Methods 1 and 3 (Cohen's d = -0.59 and -1.93, respectively) and GPT-3.5 with Method 1 (Cohen's d = -0.82). In contrast, summary and total perplexity metrics generally exhibit non-significant differences between correct and incorrect predictions.

Method 2 (ensemble) shows weaker perplexity-performance associations for GPT-3.5, likely due to the averaging effect of multiple predictions reducing individual prediction uncertainty. Method 3 exhibits the strongest perplexity signals with GPT-4o, suggesting that few-shot learning creates more calibrated confidence estimates in advanced models.

Table 9. Results of the statistical tests investigating differences in perplexity metrics between correctly and incorrectly classified issue reports.

LLM	Method	Metric	p-value	Cohen's d	Effect Size
GPT-3.5	Method 1	class perplexity	**3.15E-04**	-0.82	large
GPT-3.5	Method 1	summary perplexity	5.02E-01	–	–
GPT-3.5	Method 1	total perplexity	3.84E-01	–	–
GPT-4o	Method 1	class perplexity	**3.09E-03**	-0.59	medium
GPT-4o	Method 1	summary perplexity	9.94E-01	–	–
GPT-4o	Method 1	total perplexity	8.83E-01	–	–
GPT-3.5	Method 2	class perplexity	5.00E-01	–	–
GPT-3.5	Method 2	summary perplexity	5.11E-01	–	–
GPT-3.5	Method 2	total perplexity	6.15E-01	–	–
GPT-4o	Method 2	class perplexity	**6.00E-04**	-0.85	large
GPT-4o	Method 2	summary perplexity	9.49E-01	–	–
GPT-4o	Method 2	total perplexity	7.56E-01	–	–
GPT-3.5	Method 3	class perplexity	4.50E-01	–	–
GPT-3.5	Method 3	summary perplexity	4.98E-01	–	–
GPT-3.5	Method 3	total perplexity	4.86E-01	-	–
GPT-4o	Method 3	class perplexity	**6.59E-08**	-1.93	large
GPT-4o	Method 3	summary perplexity	6.26E-01	–	–
GPT-4o	Method 3	total perplexity	**9.73E-08**	-1.87	large

Table 10. Method 2 classification performance with alternative, perplexity-based voting mechanisms.

		F1 Measure				
LLM	Voting Mechanism	Bug	Docum.	Feature	Question	Weighted Average
GPT-3.5	majority	0.90	0.80	0.85	0.83	0.85
GPT-3.5	min class perplexity	0.91	0.82	0.86	0.84	0.86
GPT-4o	majority	0.92	0.91	0.92	0.93	0.92
GPT-4o	min class perplexity	0.92	0.91	0.92	0.93	0.92

Novel perplexity-based ensemble methods. Building on these findings, we evaluated whether replacing majority voting with minimum class perplexity voting could improve Method 2's performance (Table 10).

For GPT-3.5, perplexity-based voting provides modest but consistent improvements across all classes, yielding a +1% overall improvement in weighted F1-score (0.85 → 0.86). However, GPT-4o shows no improvement from perplexity-based voting, maintaining identical performance (0.92 weighted F1-score) regardless of voting mechanism. These results demonstrate that perplexity-based confidence estimation can enhance ensemble performance for less capable models. In contrast, advanced models may have already reached performance ceilings where sophisticated voting mechanisms provide diminishing returns.

RQ1 - Summary and Implications. The evaluation demonstrates that the proposed methods outperform existing baselines, with Method 3 reaching state-of-the-art performance (0.95 weighted F1-score). More intriguingly, Qwen-2.5 consistently outperforms the larger GPT-3.5 model, suggesting that architectural design trumps parameter count in classification tasks.

The methods reveal unexpected trade-offs and behaviors. Intent extraction provides improvements for weaker models (up to 18% for GPT-3.5) while offering diminishing returns for advanced architectures. The MapReduce approach proves essential for long issues, delivering F1-score gains (2-18 percentage points) at reasonable cost premiums (2-2.5 times more). Most surprisingly, temperature sensitivity varies dramatically by model-method combination, with some configurations showing complete performance collapse under stochastic conditions. The analysis uncovers that perplexity metrics emerge as reliable confidence predictors for ensemble optimization. Given these results, we can answer RQ1 as follows:

> **RQ1 Answer:**
> *The proposed LLM-based methods outperform existing approaches at reasonable costs, revealing that open weights models can be as performant as larger, proprietary models. The MapReduce decomposition strategy we proposed proved to be consistently more effective than both truncation and direct ingestion of full-length issue descriptions, confirming its value in preserving classification performance for long and complex reports. In addition, perplexity emerged as a reliable confidence predictor, especially for ensemble optimization.*

Based on the insights gathered, our results provide actionable implications for the design and deployment of LLM-based issue report classification systems:

- **Consider model recency and training characteristics alongside size when selecting models.** The results show that newer models with fewer parameters can outperform older, larger models, likely due to improved architectures, more recent training data, and refined training methodologies. Practitioners should evaluate modern open-weight models rather than assuming that parameter count or proprietary models necessarily correlate with task performance.
- **Use MapReduce-style summarization to enhance classification robustness on long issue reports.** Developers of AI-assisted issue triaging tools may adopt segmentation and consolidation strategies instead of relying on truncation or full-text ingestion, particularly when classifying verbose reports. Recent literature [2, 43] highlights how LLM performance tends to degrade with increasing context length—an effect we also observed in our evaluation. For GPT-4o, the only model able to fit the long issues in full, classification performance was still lower when ingesting the entire input than when using the MapReduce-based approach. In this sense, future classification frameworks might embed adaptive preprocessing components that automatically segment, reduce, and recompose lengthy issue texts to maximize interpretability while controlling token budgets and inference cost.
- **Leverage perplexity scores to enhance prediction reliability in ensemble settings.** Perplexity can be directly exploited to design smarter ensemble aggregation strategies beyond majority voting. Practitioners can incorporate confidence-aware heuristics or thresholds to filter low-certainty predictions, enabling more robust classifications in production scenarios. Hence, future ensemble-based classifiers might integrate perplexity-aware scoring layers to prioritize predictions with higher confidence, or automatically trigger abstention and fallback logic for uncertain cases.
- **Tune temperature carefully, especially in non-deterministic deployments.** Given that performance varies substantially across model-method combinations, and that higher

Table 11. Consistency Results

Method	GPT-3.5		Qwen-2.5		GPT-4o	
	Raw Perc. Agreement	Fleiss' Kappa	Raw Perc. Agreement	Fleiss' Kappa	Raw Perc. Agreement	Fleiss' Kappa
Method 1	0.95	0.93	1.00	1.00	0.99	0.99
Method 2	1.00	1.00	1.00	1.00	1.00	0.99
Method 3	0.98	0.97	1.00	1.00	0.99	0.99

temperatures may lead to instability, developers integrating LLMs may calibrate temperature settings per use case and method, or consider temperature-agnostic ensemble methods to mitigate unpredictability. As a consequence, future pipelines might include automated temperature sensitivity analysis during evaluation, or adopt hybrid settings that combine deterministic prompting with selective sampling for ambiguous cases.

5.2 RQ2: Classification Consistency

Table 11 presents the results of the classification consistency analyses, measuring raw percentage agreement and Fleiss' Kappa index across five runs for each method-LLM combination.

All methods achieve near-perfect consistency [20, 46].

Method 2 demonstrates the highest consistency, achieving perfect agreement (1.00) with GPT-3.5 and Qwen-2.5, confirming that ensemble voting effectively reduces non-deterministic variability. Notably, Qwen-2.5 achieves perfect consistency across all methods, while GPT-4o shows minimal variability (0.99 agreement). GPT-3.5 exhibits the most variability, particularly with Method 1 (0.95 raw percentage agreement), but still maintains high consistency levels.

Table 12 reveals surprising model-method interactions under different temperature settings. GPT-4o maintains stability across all temperatures and methods, with consistency never dropping below 0.93 Fleiss' Kappa, demonstrating robust architecture design for stochastic sampling.

As for Qwen-2.5, it maintains higher consistency (0.90-1.00 Kappa) than GPT-3.5 in Methods 1 and 2, across all temperatures. Nonetheless, the results show that Qwen-2.5 exhibits a severe consistency degradation with Method 3 at any temperature above 0.0, mirroring classification performance degradation also observed in Section 4.4.1. In that case, Fleiss' Kappa plummets from perfect agreement (1.00) to 0.73-0.76, representing an unanticipated architectural incompatibility between Qwen-2.5's decoder-only structure and few-shot learning under stochastic conditions.

Method 2 consistently outperforms Method 1 in consistency across all temperature-model combinations, confirming that ensemble voting effectively stabilizes predictions even under stochastic sampling. GPT-3.5 shows the most substantial ensemble benefits, with Method 2 maintaining 0.86 Kappa at T=1.0 compared to Method 1's 0.82.

RQ2 - Summary and Implications. The results discussed above lead us to answer RQ2 as follows:

> **RQ2 Answer:**
> *Classification consistency varies by model architecture and method combination, with all approaches achieving near-perfect consistency at temperature=0.0 and exhibiting only slight degradation up to temperature=0.7. While models from the GPT family maintain consistency*

Table 12. Consistency Results for the proposed methods, across different temperature settings.

Method	Temp.	GPT-3.5		Qwen-2.5		GPT-4o	
		Raw Perc. Agreement	Fleiss' Kappa	Raw Perc. Agreement	Fleiss' Kappa	Raw Perc. Agreement	Fleiss' Kappa
Method 1	0.0	0.95	0.93	1.00	1.00	0.99	0.99
	0.3	0.94	0.92	0.95	0.94	0.98	0.97
	0.7	0.91	0.87	0.94	0.92	0.97	0.97
	1.0	0.88	0.82	0.93	0.90	0.96	0.95
Method 2	0.0	1.00	1.00	1.00	1.00	1.00	0.99
	0.3	0.95	0.94	0.96	0.95	0.98	0.98
	0.7	0.93	0.89	0.95	0.94	0.98	0.97
	1.0	0.90	0.86	0.94	0.92	0.97	0.96
Method 3	0.0	0.98	0.97	1.00	1.00	0.99	0.99
	0.3	0.98	0.97	0.84	0.76	0.98	0.97
	0.7	0.97	0.95	0.82	0.73	0.97	0.96
	1.0	0.95	0.93	0.82	0.73	0.95	0.93

across all methods and temperatures, Qwen-2.5, unexpectedly, exhibits a critical incompatibility with few-shot learning when temperature > 0. Last, ensemble methods provide consistent stabilization benefits across all LLMs and temperatures.

These results carry several practical implications for researchers and practitioners developing LLM-based classification tools:

- **Avoid assuming determinism, even at temperature zero.** While temperature=0 yields near-perfect consistency across methods, minor stochastic effects are still present. This suggests that full determinism cannot be guaranteed even in supposedly stable settings. In this sense, a first implication of our results concerns the need to incorporate multiple-run evaluations and statistical agreement metrics, e.g., Fleiss' Kappa, into testing workflows for reproducibility assurance, particularly in safety- or audit-critical environments. This would be particularly relevant for researchers evaluating LLM-based classifiers in benchmark settings, where conclusions drawn from single-run metrics might be misleading. More in general, this observation may possibly apply to other classification tasks beyond issue reporting, and urges the community to treat "temperature=0" not as a guarantee of functional determinism, but rather as a high-consistency approximation with residual variance worth measuring.
- **Favor ensemble-based strategies in production environments requiring stability.** According to the results, Method 2 (ensemble voting) consistently reduces prediction variability across LLMs and temperature configurations, outperforming both zero-shot and few-shot prompting. GPT-3.5, in particular, benefits substantially from ensemble stabilization. This suggests that ensembles may act as a reliability amplifier for less robust models, allowing developers to offset model limitations through methodological design. For instance, developers may adopt ensemble-based approaches as a default when integrating LLMs into triaging or prioritization pipelines where consistent behavior across invocations is essential. At the same time, researchers conducting empirical studies might include ensemble baselines in their evaluations to isolate gains derived from aggregation mechanisms rather than LLM capabilities alone.

- **Validate compatibility between LLMs and prompting strategies before deployment.** The results reveal an architectural incompatibility between Qwen-2.5 and few-shot learning under stochastic sampling: performance and consistency both degrade sharply at $T > 0$. As such, practitioners adopting open-weight models may need to test specific model–method pairings across temperature levels before deployment, rather than assuming strategy portability across LLM families. Similarly, these findings suggest that prompt engineering guidelines may need to be architecture-specific. Researchers working on prompting techniques may benefit from reporting not only average performance but also interaction effects between prompting strategies, temperatures, and model architectures, especially for LLMs not belonging to the GPT family.
- **Include consistency as an evaluation criterion in LLM pipeline design.** While performance metrics remain the dominant focus, consistency may be considered equally important in high-reliability settings. As such, classification pipelines might benefit from consistency-focused regression testing, where repeated invocations with fixed inputs are checked for agreement, to detect fragile or temperature-sensitive configurations early in development and testing cycles. Again, these findings possibly inform the way researchers evaluate LLM-based classification systems: in addition to standard performance metrics, future work might include consistency measurements across temperature sweeps and multiple seeds, especially when targeting reproducible behavior in developer-facing tools or infrastructure components.

5.3 RQ3: Error Patterns and Classification Dynamics

This section shows the results for the analyses described in Section 4.4.3.

5.3.1 Class-level Analysis of Misclassifications. The results of the quantitative class-level analysis of misclassification are reported in Figure 14.

The figure presents confusion matrices illustrating the distribution of the incorrect predictions made by the three considered methods using GPT-3.5, Qwen-2.5, and GPT-4o. Each sub-matrix corresponds to a specific method and LLM variant and shows how many issue reports from an actual class (y-axis) were misclassified as different categories (x-axis). The color intensity of each cell represents the frequency of misclassifications, with darker shades indicating higher misclassification counts.

The analysis of classification errors reveals distinct patterns in how LLMs interpret and apply the hierarchical decision rules specified in the classification prompts. A first notable trend in GPT-3.5 is the frequency of misclassification of *question* issues across all proposed methods. In Method 1, 17 questions are misclassified as bugs (9) or features (8). In Method 3, the overall number of misclassifications of *question* issues is 12, with 11 of these cases involving confusion with the *bug* label. This suggests that the few-shot examples provided in Method 3 are effective in reducing confusion between *question* and *feature* issues. The second most challenging class for GPT-3.5 is *documentation*, which is frequently confused with the *feature* class across all proposed methods.

Qwen-2.5, in contrast, exhibits a more balanced distribution of misclassifications across classes. Still, the confusion between *bug* and *question* remains one of the most recurrent error patterns. In both Method 1 and Method 2, Qwen-2.5 misclassifies 4 *question* issues as *bugs*, while 2 bug reports are incorrectly labeled as questions. Method 3 proves effective in improving classification performance on *question* issues, reducing misclassifications to just one case, which is, again, incorrectly labeled as a bug. However, this improvement comes at the cost of increased misclassification of bug reports: 4 of them are erroneously classified as questions. This trade-off suggests that while few-shot learning enhances Qwen-2.5's ability to recognize interrogative intent, it may also lead to overgeneralization in distinguishing between technical problem descriptions and user inquiries.

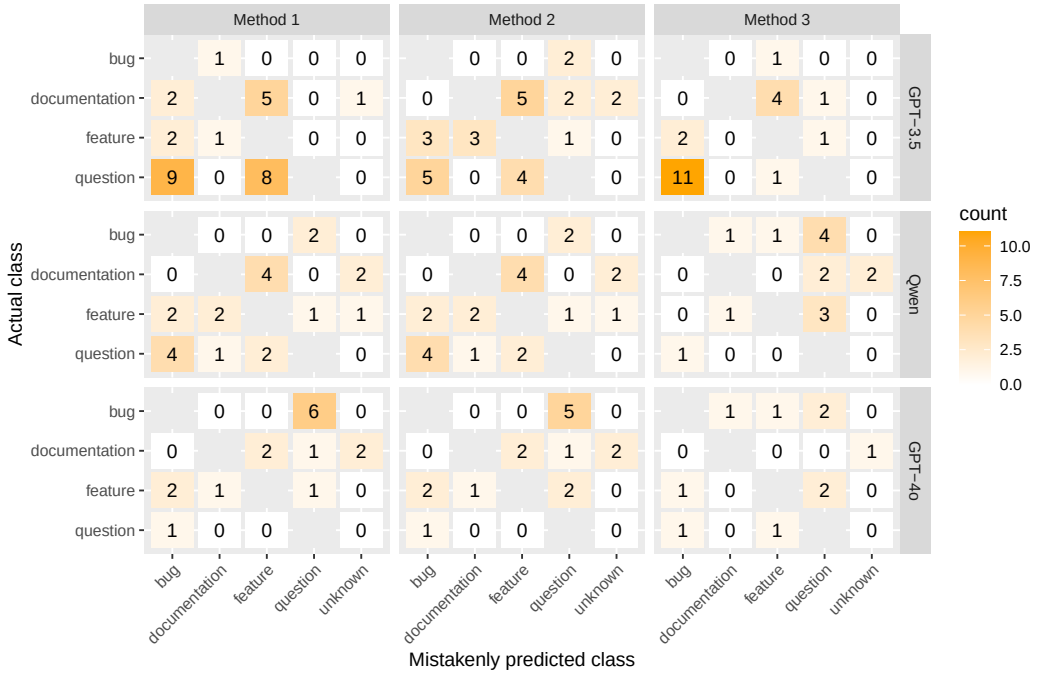

Fig. 14. Confusion Matrices of Issue Misclassifications

GPT-4o demonstrates the highest overall classification performance among the evaluated models. Its most prominent class confusion in Method 1 and Method 2 involves the misclassification of bug reports as questions, with 6 and 5 such cases observed in each method, respectively. Method 3 partially alleviates this issue, reducing the number of bug-to-question misclassifications to just 2. Nonetheless, it introduces new errors, misclassifying one bug report as documentation and another as a feature. These findings suggest that while few-shot learning improves GPT-4o's ability to distinguish between interrogative and technical content, it may also introduce subtle shifts in classification boundaries that affect precision in edge cases.

To complement the static view offered by the confusion matrices, we conducted a dynamic flow analysis of classifications using Sankey diagrams (Figure 15). This visualization captures how both correct and incorrect classifications change across the three LLMs for each method, offering a more granular understanding of classification dynamics. The x-axis represents the sequence of models: Original Label → GPT-3.5 → Qwen-2.5 → GPT-4o. Flows between different classes indicate a change in classification, while flows that remain within the same class represent stability.

The diagram confirms a clear performance hierarchy: GPT-3.5 introduces the highest number of misclassifications, Qwen-2.5 partially mitigates these errors, and GPT-4o achieves the most consistent improvements. Better-performing models tend to either correct prior misclassifications or confirm them, while transformations into entirely new errors are relatively rare.

For example, in Method 1, GPT-3.5 frequently misclassified *questions* as *features* or *bugs*. Qwen-2.5 corrected more than half of these errors, and GPT-4o further refined the predictions, resolving approximately half of Qwen-2.5's remaining errors. A similar trend also emerges for Method 2 and Method 3.

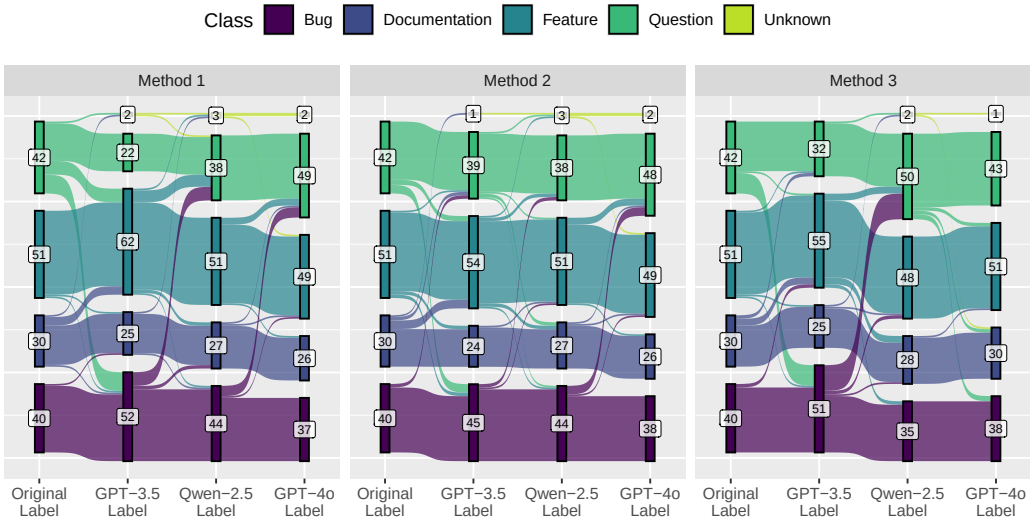

Fig. 15. Sankey Diagrams of Classification Flows Across LLMs and Methods. Each flow represents how issue classes evolve from the original label through GPT-3.5, Qwen-2.5, and GPT-4o.

While stronger models generally correct misclassifications introduced by weaker ones or confirm them when unresolved, there are notable exceptions where errors emerge for the first time in more capable models. For instance, in Method 1 and Method 2, GPT-4o introduces 4 new misclassifications of bug reports as questions, errors not present in earlier GPT-3.5 and Qwen-2.5. Similarly, in Method 3, Qwen-2.5 corrects many misclassifications introduced by GPT-3.5, particularly cases where questions were incorrectly labeled as bugs, but in doing so, it also introduces new mistakes by misclassifying some actual bugs as questions. These patterns highlight that improvements in overall performance do not entirely eliminate the emergence of novel classification mistakes.

5.3.2 Misclassification Causes. For what concerns the qualitative analysis of misclassification causes, a set of recurring misclassification drivers emerged from the process described in Section 4.4.3. These drivers were subsequently formalized into the taxonomy presented below, which captures the most salient sources of classification disagreement observed. During the independent annotation phase, the inter-rater agreement, measured using Cohen's Kappa, reached a substantial level of agreement, with a Kappa score of **0.92**, indicating that the taxonomy is well-defined and reliably applicable. As a result of the analysis process, five primary causes of misclassifications were identified, namely *class overlap*, *ambiguity*, *complexity*, *implicit intent*, and *other*, detailed as follows.

Class overlap. Some issues exhibit characteristics that align with multiple classes, making it more challenging for classification approaches to understand the primary intent of the issue and to assign a single definitive label. For example, consider an issue asking if a certain feature is available and then stating that, if not already available, that feature should be implemented. The issue has a primary questioning intent and should thus be classified as a *question*, but also exhibits overlaps with the *feature* label, as it is (even though conditionally) proposing a new feature as well. Such cases blur the boundaries between classes, increasing the likelihood of misclassification. One such case is reported in Example 2. Each example issue report in our paper is presented in a structured

format. The top bar contains the issue title, followed by the unique issue identifier from the dataset in parentheses. Below this, the main body of the issue report is displayed, preserving the original text.

Example Issue Report 2: Does MPRIS expose Genre (#1260904378)

Does MPRIS expose Genre? If so it'd be nice to be able to filter on genre,

Ambiguity. Ambiguous issue reports lack sufficient detail, contain vague or misleading language. Consider, for example, the issue report in Example 3.

Example Issue Report 3: Cannot declare class GoogleClient (#1122657441)

Hi, I want to report a bug.
since it is being used by many people, it creates conflicts when two different organizations use the same thing.
For example: I have created a plugin that uses your Social Login integration method and a separate plugin that uses your other services.
Now, the conflict occurs when we try to attempt the class in two separate plugins. Cannot declare class GoogleClient from `/google-client/vendor/google/apiclient/src/Client.php` on line 50
Is there any way to counter this?

Despite the intent of the user creating the issue report is to inquire about ways to address the naming conflict they are experiencing, the issue report starts, incorrectly, with the declaration "*I want to report a bug*". This is also sometimes due to the use of incorrect issue templates, as shown in Example 4, where a user reports a bug, but leaves the default template for asking questions on top of the issue report.

Example Issue Report 4: Multiple urls sharing not working on android (#1120539238)

Ask your Question
<!- Ask your question ->
sharing multiple audio files (array of URLs) works fine on ios but on android, its only shares the first URL not all the URLs"

Ambiguity increases the likelihood of incorrect predictions, particularly with simpler classification models that rely on surface-level cues rather than deeper contextual understanding.

Complexity. Some misclassified issues are inherently more complex, exhibiting long, technical descriptions (including snippets of code or execution logs), and possibly the involvement of multiple concerns into a single issue. Such cases may require reasoning beyond what the model is capable of handling, leading to errors. Complexity-related misclassifications often involve confusion between `question` and bug, as some issues start by describing unintended behavior or errors the users are experiencing, and then proceed to ask a question on how to properly use the software. As an example, consider the issue in Example 5.
The above issue is classified as a question, as the primary intent is to acquire feedback on how to fix or prevent the described error. However, the complexity and technical details in the first part of the issue body lead models to concentrate their attention on the first part, and incorrectly classify the issue as a bug.

Example Issue Report 5: Cannot add a new users schema extension through "UserSchema.json" (#1372841808)

Hello, I got back a 400 ERROR while sending a POST request to "/Users" with the following request body

```
{
    "schemas": [
        "urn:ietf:params:scim:schemas:core:2.0:User",
        "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User"
    ],
    "externalId": "external",
    "userName": "name@name.com",
    "meta": {
        "resourceType": "User"
    },
}
```

this is because this schema "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User" was not defined in the "UserSchema.json" file. So, once I added it manually to the database table "SCIMSchemaExtension" it worked just fine. So, can you please tell me how can I fix this? Thanks a lot

Implicit intent. Some issue reports contain subtle nuances that may be more difficult for the model to interpret, even though the issue description itself is neither particularly complex nor ambiguous. For these issues, correct classification depends on the model's ability to *read between the lines* and capture implicit intent from subtle contextual clues. To better explain this root cause of misclassification, a representative case is shown in Example 6.

Example Issue Report 6: OTP produces no meaningful message when GTFS input is corrupt (#29698056)

Reported on mailing list: https://groups.google.com/d/msg/opentripplanner-users/otLwIZQOTpQ/9LKYdz4IEzYJ
Corrupted ZIP file just yields cryptic "no agencies" error.

Even though the report does not explicitly request a change to an existing feature, it implicitly suggests an evolutionary maintenance action, i.e., to improve an existing feature by providing a more informative error message when incurring in invalid inputs. Nonetheless, the models may erroneously classify the issue as a bug, since it explicitly describes an undesired behavior rather than directly proposing an enhancement.

Other. This category was assigned to misclassified issues that do not fit any of the other categories and appear to be misclassified for no clear reason. These instances could not be attributed to a specific cause, possibly indicating inherent model limitations. For instance, consider the issue report in Example 7. The issue report's main intent is to inquire about how to enable a certain functionality,

Example Issue Report 7: How to enable "auto format" and "auto insert row" functionality? (#1143694432)

Tab doesn't do anything for me by default. I couldn't find any references to such settings in this repo, the only places where the word `auto` comes up are README files and doc strings.

and thus it is labeled as a *question* in the dataset. However, GPT-3.5 with Method 1 labels the issue as a *feature* request, despite the absence of an explicit suggestion for new functionality.

Table 13 reports the distribution of misclassification causes across models and LLMs. Each row of the table corresponds to a specific misclassification cause we defined in the proposed taxonomy, while columns indicate how many times a given cause was observed for a given method and LLM.

Table 13. Distribution of Misclassification Causes Across Methods and LLMs. For the sake of conciseness, Method 1, Method 2, and Method 3 are abbreviated as M1, M2, and M3, respectively.

	GPT-3.5			Qwen-2.5			GPT-4o		
	M1	**M2**	**M3**	**M1**	**M2**	**M3**	**M1**	**M2**	**M3**
Class overlap	3	3	0	2	2	1	2	2	0
Ambiguity	4	5	3	5	5	4	4	4	3
Complexity	4	1	6	1	1	2	0	0	0
Implicit intent	5	6	5	4	4	3	3	3	2
Other	13	12	7	9	9	5	7	7	5
Total	**29**	**27**	**21**	**21**	**21**	**15**	**16**	**16**	**10**

Among the identified causes, *other*, which refers to errors without a clear explanation, appears to be the most frequent, accounting for approximately half of the misclassifications across all models and methods. Ambiguity emerges as a common misclassification cause, affecting all the considered LLMs in a comparable way. This indicates that insufficient or vague issue descriptions remain a persistent challenge, regardless of model size or architecture. *Complexity*, on the other hand, disproportionately affects GPT-3.5, particularly in Method 3, where six such cases are recorded. This suggests that GPT-3.5 struggles more with long, technical issue reports involving multiple concerns, while GPT-4o and Qwen-2.5 appear more resilient to higher input complexity. *Class overlap* is another notable source of misclassification, especially in Method 1 and Method 2 across all models. Interestingly, Method 3, leveraging few-shot examples, is effective in mitigating this issue, eliminating class overlap errors entirely for GPT-3.5 and GPT-4o, and reducing them to just one case for Qwen-2.5. This highlights the potential of example-based prompting in helping models disambiguate overlapping intents. Finally, *Implicit intent* remains a challenge across all models, though it is more prevalent in GPT-3.5. GPT-4o, while not immune, shows a marked reduction of approximately 50% in such errors across all methods. Qwen-2.5 shows intermediate behavior, with fewer such errors than GPT-3.5 but slightly more than GPT-4o, suggesting it captures implicit signals reasonably well, though not yet at the level of GPT-4o.

5.3.3 Combined analysis. In this section, we combine the quantitative distribution of class-level misclassification with the qualitative taxonomy of their underlying causes, highlighting how different LLMs and classification methods interact with specific challenges. The bar chart in Figure 16 provides a consolidated view of how different misclassification causes are distributed across models and methods and specific class confusions, offering a deeper understanding of the interplay between classification strategies and LLM behavior.

The most frequent confusion, i.e., the one between *question* and *bug*, is strongly associated with causes like *implicit intent* and *complexity*, particularly for GPT-3.5 and Qwen-2.5 in Methods 1 and 2. These models exhibit a tendency to misinterpret technically detailed inquiries as bug reports, especially when the user's intent is not explicitly stated. Conversely, GPT-4o, while more robust to *complexity*, still shows a tendency to misclassify *bug* reports as *questions*, especially in Methods 1 and 2, where clarification-seeking language is present. This reflects a heightened sensitivity to conversational cues, which can override procedural context. The confusion between *documentation* and *feature* is another recurring pattern, often driven by *ambiguity* and *class overlap*, particularly in Qwen-2.5 and GPT-3.5. Method 3 generally reduces these errors, especially those due to *ambiguity*, but introduces new ones, such as *bug* to *question* misclassifications, likely due to the influence of

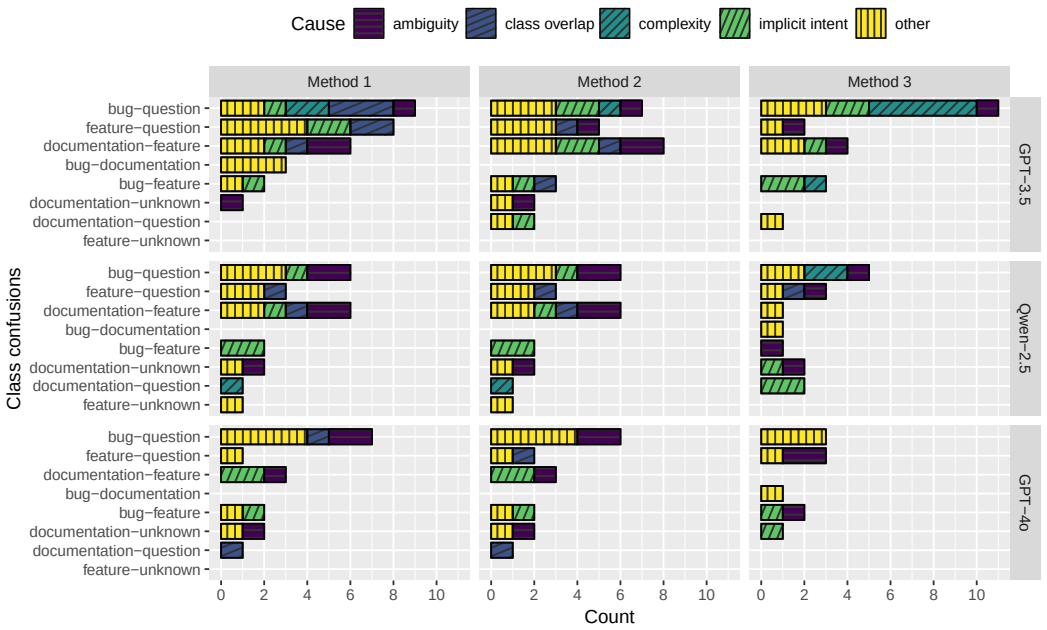

Fig. 16. Combined analysis of classification confusions and misclassification causes across methods and LLMs.

few-shot examples that overemphasize inquiry patterns. Overall, this analysis confirms that, while Method 3 enhances robustness against certain misclassification causes, particularly *ambiguity* and *class overlap*, it also shifts the error landscape by amplifying sensitivity to *implicit intent*. Each model exhibits distinct interpretive biases: GPT-3.5 struggles with complexity, Qwen with class overlap, and GPT-4o with subtle intent cues. These complementary strengths and weaknesses suggest promising opportunities for hybrid strategies that integrate multiple methods or models, potentially through ensemble architectures that adaptively leverage their respective advantages.

5.3.4 Cross-Model and Method Comparative Analysis. Comparing the three classification methods across all LLMs reveals complex performance dynamics, with error patterns that shed light on the efficacy of the methods, model behavior, and their interactions with varying informative characteristics of the reports. In the following, in order to systematically examine performance dynamics, we analyze error patterns across six key dimensions, which together allow us to define a recovery strategy for misclassification. Unless otherwise noted, in this subsection, a *partial failure* denotes an issue for which at least one LLM predicted an incorrect label, whereas a *complete failure* denotes issues misclassified by all three LLMs.

Error Footprint and Intersections Across Methods. To characterize the overlap and divergence of misclassification patterns across methods, Table 14 reports the number of partial and complete failures for each method, together with their intersections and relative complements. These figures highlight how errors cluster across approaches and where unique or shared weaknesses emerge.

Two key findings emerge when examining how different methods modify the error landscape. First, **ensemble consolidation**: rather than correcting Method 1's mistakes, Method 2 tends to reinforce them. Out of the 43 issue misclassification cases of Method 1, 33 remain incorrect also in

Table 14. Cross-method error footprint and overlaps. Method names are abbreviated as M1 (Intent Extraction), M2 (Ensemble), and M3 (Explained Few-Shot).

				Intersections				**Relative Complements**					
	M1	M2	M3	M1∩M2	M1∩M3	M2∩M3	M1∩M2∩M3	M1\M2	M2\M1	M1\M3	M3\M1	M2\M3	M3\M2
Partial failures	43	38	33	33	21	19	18	10	5	22	12	19	14
Complete failures	8	11	3	8	2	2	2	–	3	6	1	9	1

Table 15. Error counts by token length (title+body). Method names are abbreviated as M1 (Intent Extraction), M2 (Ensemble), and M3 (Explained Few-Shot).

	Partial Failures			**Complete Failures**		
Bucket (tokens)	**M1**	**M2**	**M3**	**M1**	**M2**	**M3**
0–50	9	10	6	3	4	0
51–90	8	8	8	3	3	1
91–150	6	8	6	0	1	1
151–500	18	12	2	2	3	1
≥ 500	2	0	11	0	0	0
Total	**43**	**38**	**33**	**8**	**11**	**3**
Share < 500	**41/43**	**38/38**	**22/33**	**8/8**	**11/11**	**3/3**
Mean tokens (error set)	**200**	**137**	**268**	**108**	**122**	**144**

Method 2 ($M1 \cap M2$). Moreover, the ensemble even introduces 5 new errors in issues that were classified correctly by Method 1 ($M2 \setminus M1$). This ensemble error consolidation effect also manifests in the complete failures error category. While Method 1 exhibits 8 cases where all LLMs misclassify, Method 2 increases this number to 11. This rise reinforces the interpretation that majority voting over homogeneous pipelines does not mitigate systematic weaknesses; instead, it amplifies them when all ensemble members share similar blind spots. In other words, aggregation without diversity hardens correlated errors, turning isolated misclassifications into consensus failures.

Second, **few-shot learning trade-offs**: Method 3 demonstrates a more transformative effect, recovering 22 of Method 1's errors ($M1 \setminus M3$) and reducing complete failures from 8 to 3. However, this improvement comes at a cost: 12 issues that were correctly classified by Method 1 become misclassified ($M3 \setminus M1$). These shifts reveal that example-guided reasoning alters the decision boundary, often improving coverage of difficult cases while redistributing errors elsewhere.

Informational Quality and Classification Robustness. Dimensional analysis shows how the length of issue reports in terms of tokens influences classification performance in non-linear ways. Table 15 reports the distribution of partial and complete failures by token length, considering the total token count (title plus body) of the report. The results show that misclassifications are disproportionally concentrated within shorter issue reports: although issues shorter than 90 tokens constitute only about 35% of the test set (see Figure 12), they account for roughly 65% of all complete failures and

Table 16. Error counts by technical information (stack traces/logs) and token length. Method names are abbreviated as M1 (Intent Extraction), M2 (Ensemble), and M3 (Explained Few-Shot).

		Partial Fail.			Complete Fail.		
Technical information	**Token length bucket**	**M1**	**M2**	**M3**	**M1**	**M2**	**M3**
With logs and/or stack traces	0–50	0	0	0	0	0	0
	51–90	0	0	0	0	0	0
	91–150	0	0	0	0	0	0
	151–500	2	1	2	0	0	0
	≥ 500	2	0	3	0	0	0
	Total	**4**	**1**	**5**	**0**	**0**	**0**
Without logs/stack traces	0–50	9	10	4	3	4	0
	51–90	8	8	4	3	3	1
	91–150	6	8	6	0	1	1
	151–500	16	11	13	2	3	1
	≥ 500	0	0	1	0	0	0
	Total	**39**	**37**	**28**	**8**	**11**	**3**
Grand Total		**43**	**38**	**33**	**8**	**11**	**3**

45% of partial failures, indicating that informational scarcity is a key driver of misclassification. Indeed, as shown in the table, in Method 1, 41 out of 43 partial failure cases occur below 500 tokens, with 23 cases below 151 tokens and a clear concentration in minimal-information ranges: 9 cases (0-50 tokens), 8 cases (51-90 tokens), and 6 cases (91-150 tokens). This trend is consistent with errors driven by *ambiguity* and missing context (Table 13), and can be observed also in Method 2, where all partial failures occur on issues shorter than 500 tokens. In Method 3, a similar trend occurs, with 22 out of 33 misclassified issues containing fewer than 500 tokens and 14 cases containing fewer than 151 tokens. Complete failures are even more strongly associated with short reports. For both Methods 1 and 2, all complete failures occur exclusively below 500 tokens, with 65% of them concentrated below 90 tokens. Even for Method 3, which substantially reduces the number of complete failures overall, all remaining complete failures still appear in issue reports shorter than 500 tokens. These results indicate that minimal textual input severely limits classification robustness, independently of the prompting strategy used.

To gain additional insights on the role of informational richness on misclassifications, we further analyzed failures by explicitly distinguishing issues with and without technical artifacts such as logs or stack traces. Table 16 reports failure counts conditioned on both token length and the presence of technical information. The results show a clear separation: misclassifications are overwhelmingly concentrated in issues that do not contain technical artifacts. Across all methods, no complete failures occur in issues that include logs or stack traces, regardless of their length. As for partial failures, they overwhelmingly originate from issue reports that do not contain technical artifacts. As shown in Table 16, 39 out of 43 partial failures for Method 1 occur in artifact-free issues, with similarly skewed distributions for Methods 2 and 3. This confirms that informational quality, rather than token length per se, is a critical factor affecting classification robustness.

Finally, when focusing specifically on artifact-free issues, we observe model-dependent behaviors. As reported in Table 17, GPT-based models (GPT-3.5 and GPT-4o) exhibit consistent trends across

Table 17. Breakdown of failure counts by method, model, and token length for issues with no technical information. Method names are abbreviated as M1 (Intent Extraction), M2 (Ensemble), and M3 (Explained Few-Shot).

	GPT-3.5			Qwen-2.5			GPT-4o		
Token length bucket	**M1**	**M2**	**M3**	**M1**	**M2**	**M3**	**M1**	**M2**	**M3**
0–50	6	8	1	7	7	3	5	5	3
51–90	5	6	3	6	6	2	3	3	1
91–150	3	5	5	4	4	3	3	3	1
151–500	11	7	6	6	6	9	5	6	5
≥ 500	0	0	1	0	0	0	0	0	0
Total	25	26	16	23	23	17	16	17	10

Table 18. Performance–cost comparison of routing strategies and baselines, considering weighted-average F1 and average cost per issue. Privacy-sensitive baseline is Qwen-2.5 with Method 3; privacy-unconstrained baseline is GPT-4o with Method 3.

	Privacy-sensitive			Privacy-Unconstrained		
Metric	**Baseline**	**Router**	**Δ**	**Baseline**	**Router**	**Δ**
Weighted F1	0.92	0.94	+0.02	0.95	0.95	–
Cost/issue (USD)	0.003	0.003	–	0.010	0.005	−50%

methods and token lengths, with Method 3 typically yielding fewer failures. In contrast, for Qwen-2.5 this trend diverges once issue descriptions exceed 150 tokens: in the 151–500 token range, Method 1 results in fewer failures than Method 3 (6 vs. 9 cases), corresponding to a reduction of approximately 25%. This suggests that, for longer natural-language–only reports, Qwen-2.5 benefits more from intent abstraction than from example-driven reasoning, whereas GPT-based models remain more robust to the choice of prompting strategy under the same conditions.

Adaptive Routing Strategies Across Privacy Constraints. Building on the analysis presented above, we introduce an adaptive routing approach with two variants optimized for different privacy requirements. Each variant applies simple decision rules grounded in the results of the previous analyses to select the most suitable method-model combination for a given issue report, balancing accuracy, cost, and operational constraints. An overview of the proposed routing approach is presented in Figure 17.

For privacy-sensitive scenarios, where external API calls to proprietary models are not permitted, we propose routing all issues through a Qwen-2.5 instance installed on-premises. The default method is Method 3, but when an issue is text-only (no logs or stack traces) and longer than 150 tokens, we switch to Method 1. This adjustment addresses the observed vulnerability of Qwen-2.5 with Method 3 on verbose, artifact-free reports. In our test set, as reported in Table 18, this router processed 21 issues (≈ 11%) with Method 1 and the remaining 166 issues (≈ 89%) with Method 3, achieving a weighted-average F1 of 0.94, compared to 0.92 for the best privacy-sensitive pure baseline (Qwen-2.5 with Method 3 only).

For privacy-unconstrained scenarios, where external calls to proprietary LLMs are allowed, we suggest starting with Qwen-2.5 using Method 3, which proved to be the method-model combination

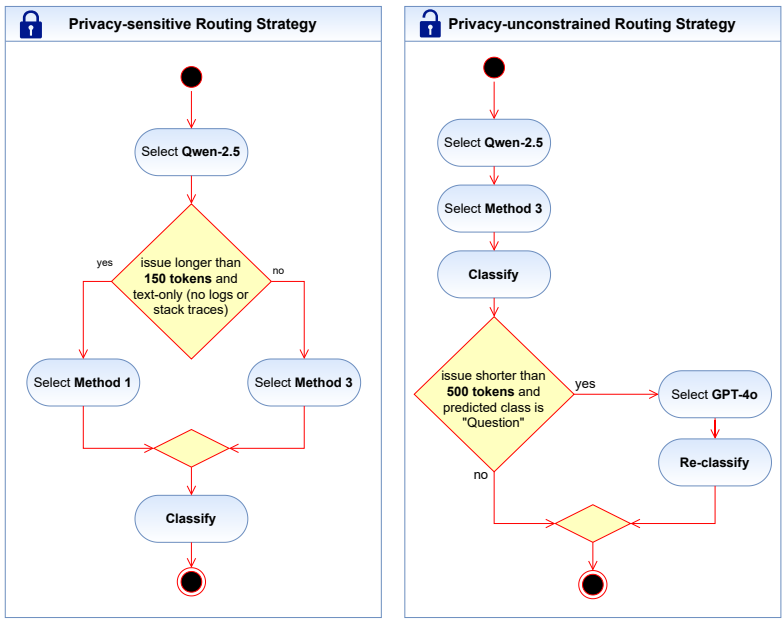

Fig. 17. Overview of the proposed Adaptive Routing Strategies for Privacy-sensitive and Privacy-unconstrained scenarios.

offering the best performance per dollar (see Section 5.1.5), for all issues. Then, escalate to GPT-4o (Method 3) only for the issues that (1) are shorter than 500 tokens, and (2) for which Qwen-2.5 (Method 3) predicted the class *question*. This pattern is often associated with Qwen-2.5 misclassifications, while GPT-4o recovers more than half of these errors (see Figure 14 and Figure 15). This routing strategy processes all issues with the cheaper Qwen-2.5 model, while reserving more expensive GPT-4o calls for a small subset of 34 issues ($\approx$ 18% of the test set), achieving, as reported in Table 18, the same weighted-averaged F1 as the best-performing GPT-4o (Method 3) baseline (0.95), but at half the average cost per issue ($ 0.005 vs $ 0.010).

RQ3 - Summary and Implications. On the basis of the results discussed above, we can address RQ3 as follows:

> **RQ3 Answer:**
> *Error patterns reveal vulnerabilities concentrated in question-bug distinctions, with ensemble methods consolidating rather than correcting mistakes. Few-shot learning creates model-specific performance trade-offs rather than universal gains, while content density below 500 tokens and lack of technical details are consistently associated with failures. These dynamics suggest complementary model architectures that could be strategically combined rather than naively aggregated.*

The results of RQ3 offer several implications for the design, deployment, and evaluation of LLM-based issue classification systems, and open concrete research directions:

- **Ensemble error consolidation** While Method 2 improves consistency (as shown in RQ2), it often fails to correct misclassifications and may even consolidate incorrect predictions (see

Table 14). This suggests that ensemble methods should be evaluated on both consistency and error profiles before being integrated into production. For researchers, this highlights an opportunity to explore error-aware or adaptive ensemble mechanisms. . While the perplexity-based voting mechanisms we proposed in Section 5.1.6 mitigate this error consolidation issue (see Table 10), future research may investigate more refined or adaptive approaches that consider heterogeneous LLMs and/or more sophisticated decision rules.

- **Impact of Informational Quality on Classification Robustness.** Our results show that most classification failures happen on short issue reports missing sufficient context details (see Table 15 and Table 16). Notably, issues shorter than 90 tokens, despite constituting only 35% of the test set, account for 65% of all complete failures and 45% of partial failures, regardless of the prompting strategy. This suggests that proactively flagging such low-information cases for human review could substantially improve end-to-end reliability at limited additional cost.
- **Adaptive strategies that leverage model-specific strengths and weaknesses.** The analysis demonstrates that both models and prompting strategies exhibit systematic, complementary strengths and weaknesses. GPT-3.5 tends to underperform on abstract or multi-label issues, Qwen-2.5 is more prone to class overlap errors, and GPT-4o can occasionally miss subtle intent cues. Likewise, prompting strategies such as intent extraction or explanatory few-shot learning show varying effectiveness depending on input characteristics like issue length, technical complexity, or ambiguity. As shown in Section 5.3.4, adaptive routing strategies that dynamically select the most suitable LLM/Method for each issue can both improve classification performance and reduce inference costs across privacy-sensitive and privacy-unconstrained deployment settings. In our privacy-unconstrained scenario, our adaptive strategy falls back on GPT-4o for a small subset of issues, specifically short reports (fewer than 500 tokens) that Qwen-2.5 systematically misclassifies as Questions. This pattern suggests a targeted opportunity for further optimization: future research could explore using specialized open-weights models, potentially fine-tuned for short or challenging issue types, to eliminate reliance on proprietary models entirely and further reduce operational costs.

5.4 Lessons Learned

We can now integrate the RQ results into a unified view that illustrates the behaviors of the three methods with respect to the operational dimensions considered and how these directly impact real-world applicability.

From the perspective of operational characteristics, the proposed methods first have different usage costs. Method 1 consumes fewer tokens and, consequently, has lower computational costs, making it particularly suitable for scenarios characterized by high volumes of issues to classify. Method 2 has a token cost that is 5 times higher on average (therefore also at the computational level), due to ensemble aggregation, but also has more predictable resource consumption patterns and results. Method 3 requires many fewer input tokens compared to Method 2, but still more than Method 1 due to few-shot examples. This introduces cost implications that scale with the volume of issues to classify. Regarding the handling of long texts, all methods benefit from the MapReduce approach. However, Method 3 shows the most marked performance improvements (up to 18 percentage points) compared to baselines, suggesting that example-based reasoning works particularly well when the report is rich in useful contextual information.

Our analysis shows that each method handles prediction confidence differently in terms of consistency and uncertainty calibration. Method 1 provides the most reliable perplexity signals for quantifying uncertainty, allowing for the identification of issue reports that need human

supervision in production settings through adaptive confidence thresholds. However, by masking the uncertainties of individual predictions, Method 2 achieves the highest consistency through aggregation, making it challenging to identify low-confidence cases that would need practitioner screening. Method 3 produces more variable perplexity signals, making it impossible to calibrate trustworthy confidence thresholds. Moreover, we found that, in Method 2, perplexity metrics can be used in more advanced ensemble approaches to replace simple majority voting, slightly improving performance (see Table 10) while preserving consistency.

Table 19. Interpretive framework of insights across RQ1–RQ3.

Method 1	Method 2	Method 3
RQ1: Method Effectiveness and Operational Characteristics		
• Abstracting to "intent" boosts weaker or older LLMs and reduces sensitivity to noisy technical details. • Use class-level perplexity as a built-in confidence signal to gate low-certainty predictions or trigger review. • It is the best cost–accuracy workhorse.	• Ensembles boost stability more than accuracy; returns diminish as base models improve. • Majority voting hardens shared biases; replace or augment it with confidence-aware rules. • Budget for the aggregation cost upfront; deploy when SLAs prioritize stability and traceability over marginal accuracy gains. • Perplexity-based ensemble voting can slightly improve classification performance on older models, while improving transparency and possibly serving as a trigger for human review.	• Examples with explanations move the decision boundary from pattern matching to rule-like reasoning—most useful where classes overlap. • Effectiveness depends on architectural fit; keep decoding deterministic and validate on the target LLM before rollout. • Accept a higher token footprint as the price of sharper class discrimination; reserve it for critical or ambiguous cases.
RQ2: Classification Consistency		
• Constraining inputs to an "intent summary" improves agreement by standardizing the decision surface; though model-dependent variability persists. • Treat temperature as a consistency budget: minimal for production, experimental only with monitoring.	• Aggregation suppresses stochastic drift; think of it as consistency insurance.	• With GPT-like models, explanations are compatible with high consistency at low temperature; with decoder-only models, explanations require stricter decoding to remain stable.
RQ3: Error Patterns and Classification Dynamics		
• Symptom-vs-intent tension drives bug-question confusions; intent extraction helps but cannot fix underspecified or very short reports.	• Ensembles herd errors when members share the same blind spot; inject heterogeneity (e.g., model diversity) and use confidence-weighted voting to counteract.	• Explanations reduce ambiguity and class overlap but can tilt decisions toward inquiry cues; monitor the bug-question boundary and calibrate exemplars accordingly. • Performance is context-elastic: it thrives with sufficient signal (richer text), but on sparse inputs the trade-offs become model-specific.
Adaptive routing strategies. Models and prompting strategies exhibit systematic, complementary strengths and weaknesses. Adaptive routing strategies can select the best LLM/Method combination for each issue, improving accuracy and/or reducing inference costs in both privacy-sensitive and privacy-unconstrained deployment settings.		

Regarding error patterns and temperature sensitivity, each method shows distinct error modes and robustness characteristics. The results demonstrate that the impact of temperature on classification performance (but generally on the task to be performed) is not predictable a priori and must be determined empirically for each method-model combination. There is no general rule that allows anticipating whether a particular classification task requires more deterministic sampling (temperature=0) or a certain degree of stochasticity, and this unpredictability represents an important aspect to consider in designing LLM-based applications.

Indeed, Method 1 has virtually stable behavior, regardless of the temperature value. Method 2 consolidates rather than corrects systematic biases, particularly when distinguishing question-bug, but has the most stable behavior as temperature varies. Method 3 has variable behavior with respect to temperature that is model-specific, and performance degrades with decoder-only architectures under non-deterministic sampling as temperature increases.

In other words, the findings reveal that operational requirements should guide method selection rather than accuracy metrics alone. Unlike traditional ML deployment, where model selection focuses primarily on predictive performance, LLM-based issue classification requires balancing different operational constraints: cost budget, consistency requirements, uncertainty quantification needs, and infrastructure capabilities. This result represents a significant shift toward "operationally aware" AI deployment in software engineering and contributes to the broader understanding of how LLMs can be strategically integrated into classification workflows, providing a framework for constraint-driven method selection in automated issue triaging systems.

Building on this perspective, our analysis of adaptive routing strategies illustrates how such operational awareness can be translated into concrete decision rules that exploit complementary strengths across LLMs. In privacy-sensitive settings, simple heuristics based on token length and the presence of technical artifacts enable the system to avoid known error patterns while remaining fully local. In privacy-unconstrained settings, selectively escalating only a small subset of short, error-prone issues to GPT-4o achieves GPT-4o-level accuracy at roughly half the cost. These results show that lightweight, empirically grounded routing policies can materially improve robustness and efficiency, reinforcing the idea that deployment-time constraints and operational characteristics should drive the design of LLM-based classification pipelines.

Table 19 synthesizes the method-specific insights across RQ1–RQ3.

6 Threats to Validity

In this section, we discuss threats that could have affected the results and their generalizability, according to the guidelines defined by Wohlin et al. [45].

A possible threat to the *internal validity* of our findings lies in the manual verification of the issue report labels in the training and test sets, which may have been influenced by annotators' bias and subjectivity of the annotators, as well as human errors. To mitigate this threat, two of the authors of this work independently annotated the issue reports, following the same procedure. After the labeling phase, any disagreements between the annotators were discussed until reaching a consensus, and any remaining ambiguous cases were discarded. This annotation approach mirrors the procedure used by Colavito et al. when preparing the original dataset [15] that we extended. Moreover, we observed very high agreement between the two independent annotators, suggesting that the annotations are reliable. Despite these precautions, the manual annotation process may still introduce bias or human errors, though we believe their impact to be minimal. An additional threat to internal validity arises from the choice of hyperparameters in the experiments. For the RoBERTa and SetFit baseline models, we used the same hyperparameters and configurations reported in prior studies on the NLBSE'23 dataset, which may not necessarily be optimal for our current setting. For the LLM-based methods, we conducted a preliminary study (described in Section 4.2) to

determine the best model configuration, favoring consistency in the classification results. While we aimed to select parameters that maximize performance, it is possible that different hyperparameter configurations could lead to better or worse results, potentially affecting the reliability of our conclusions. Another potential limitation comes from the inability to verify whether the issue reports in the test set were included in the training data for GPT-3.5, GPT-4o, and Qwen 2.5. This could have affected the validity of our results, with LLMs being influenced by pre-existing data, rather than truly generalizing to unseen examples. As a result, the observed results may not fully reflect the real-world performance of the models on new, unseen issue reports.

A possible threat to *external validity*, affecting the generalizability of our findings, arises from the fact that our results are based on a relatively small dataset of GitHub issue reports. While we used stratified sampling to ensure proper representation of all classes of issue reports, we acknowledge that the generalizability of our proposals to different types of issue is uncertain, and the results may not fully extend to other issue tracking systems, to specific domains or languages, or to issue reports with different characteristics. Further replications involving a broader set of diverse issue reports might be required to confirm the generalizability of the findings.

7 Conclusions

This study introduced two conceptual approaches to issue classification, implemented through three LLM-based methods. The first approach focuses on understanding user intent through "Intent Extraction and Classification" (Method 1) and its ensemble variant (Method 2). The second approach uses explicit reasoning with examples in the "Explained Few-Shot Learning" method (Method 3). Our evaluation demonstrates that these methods surpass existing baselines, with GPT-4o achieving the highest weighted F1-score with Method 3. Beyond performance improvements, we discovered that ensemble voting consolidates errors rather than correcting them, and few-shot learning redistributes rather than reduces mistakes. We also revealed that newer open-weight models can match or exceed the performance of older, larger proprietary models, suggesting opportunities for cost-effective deployment. Our MapReduce approach delivers advantages for long reports, while classification consistency reaches near-perfect levels across different temperature settings. Most importantly, we established a framework for understanding misclassification patterns and identified complementary strengths across model architectures that enable strategic deployment optimization. Finally, we proposed innovative ensemble techniques based on LLM perplexity and adaptive routing approaches for classifying issues. These methods are tailored for both privacy-sensitive and privacy-unconstrained contexts, focusing on enhancing classification performance while minimizing costs.

Future research could focus on expanding the dataset to include a more diverse array of issue reports to improve the generalizability of the findings. The architectural insights and behavioral patterns we discovered suggest promising directions for more sophisticated classification approaches. Additionally, exploring the integration of other LLMs may offer insights into optimizing classification performance. We also aim to refine our MapReduce-based approach to enhance scalability and efficiency, optimizing processing time and adaptive chunking strategies for improved classification performance of detailed issue reports in large-scale projects. Finally, incorporating user feedback mechanisms to refine classification models continuously could result in more adaptive and robust systems.

Another promising direction for future work involves reformulating the issue classification task as a multi-label problem. Our empirical findings reveal frequent intent overlap between categories, particularly among semantically related labels such as "support" and "enhancement". While we continue to view single-label classification as both meaningful and practically relevant, especially since many triaging systems in use today rely on assigning a single primary label for downstream

processes such as routing, prioritization, or task assignment, the observed ambiguity in issue semantics suggests that multi-label formulations could better reflect the real-world complexity of issue reports. Future research might explore suitable annotation guidelines, architectural adaptations, and evaluation protocols for multi-label classification in this context, while carefully assessing its benefits and trade-offs in terms of model performance, interpretability, and integration into existing triaging workflows. Moreover, future work could explore prompt sensitivity analysis by systematically evaluating the impact of semantically similar but differently phrased prompts on classification performance, providing additional insights into the robustness and generalizability of LLM-based classification approaches.

Acknowledgments

This work has been partially supported by the Italian PNRR MUR project PE0000013-FAIR. Also, this work has been partially supported by the *Qual-AI* national research project, which has been funded by the MUR under the PRIN 2022 program (grant number: 2022B3BP5S).

Data availability statement

Data and code required to reproduce the findings of this study are available at reference [18].

References

[1] Waleed Alhindi et al. 2023. Issue-Labeler: an ALBERT-based Jira Plugin for Issue Classification. In *2023 IEEE/ACM 10th Intern. Conf. on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE, 40–43.

[2] Chenxin An, Shansan Gong, Ming Zhong, Xingjian Zhao, Mukai Li, Jun Zhang, Lingpeng Kong, and Xipeng Qiu. 2024. L-Eval: Instituting Standardized Evaluation for Long Context Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 14388–14411.

[3] Giuliano Antoniol et al. 2008. Is it a bug or an enhancement? A text-based approach to classify change requests. In *Proc. of the 2008 Conf. of the Center for Advanced Studies on Collaborative Research*. 304–318.

[4] Gabriel Aracena, Kyle Luster, Fabio Santos, Igor Steinmacher, and Marco Aurelio Gerosa. 2024. Applying Large Language Models to Issue Classification. In *Proceedings of the Third ACM/IEEE International Workshop on NL-Based Software Engineering* (Lisbon, Portugal) *(NLBSE '24)*. Association for Computing Machinery, New York, NY, USA, 57–60. doi:10.1145/3643787.3648043

[5] Olga Baysal et al. 2013. Situational awareness: personalizing issue tracking systems. In *2013 35th Intern. Conf. on Software Engineering (ICSE)*. IEEE, 1185–1188.

[6] Nicolas Bettenburg, Sascha Just, Adrian Schröter, Cathrin Weiss, Rahul Premraj, and Thomas Zimmermann. 2008. What makes a good bug report?. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. 308–318.

[7] Tom Brown et al. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, Vol. 33. 1877–1901.

[8] Oscar Chaparro, Carlos Bernal-Cárdenas, Jing Lu, Kevin Moran, Andrian Marcus, Massimiliano Di Penta, Denys Poshyvanyk, and Vincent Ng. 2019. Assessing the quality of the steps to reproduce in bug reports. In *Proceedings of the 2019 27th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*. 86–96.

[9] Mark Chen, Jerry Tworek, Heewoo Jun, and et al. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374 [cs.LG]

[10] Songqiang Chen, Xiaoyuan Xie, Bangguo Yin, Yuanxiang Ji, Lin Chen, and Baowen Xu. 2020. Stay professional and efficient: automatically generate titles for your bug reports. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. 385–397.

[11] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, et al. 2022. Scaling Instruction-Finetuned Language Models. arXiv:2210.11416 [cs.LG]

[12] Cloudflare, Inc. 2025. Cloudflare Workers AI Pricing. https://developers.cloudflare.com/workers-ai/platform/pricing/ Accessed: 2025-07-30.

[13] Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and psychological measurement* 20, 1 (1960), 37–46.

[14] Giuseppe Colavito et al. 2022. Issue report classification using pre-trained language models. In *Proc. 1st Int. Workshop on Nat. Lang.-based Softw. Eng.* 29–32.
[15] Giuseppe Colavito et al. 2023. Few-Shot Learning for Issue Report Classification. In *Proc. of the 2023 IEEE/ACM 2nd Int. Workshop on NLBSE*. IEEE, 16–19.
[16] Giuseppe Colavito, Filippo Lanubile, Nicole Novielli, and Luigi Quaranta. 2024. Leveraging GPT-like LLMs to Automate Issue Labeling. (2024).
[17] Gabriele De Vito, Fabio Palomba, Carmine Gravino, Sergio Di Martino, and Filomena Ferrucci. 2023. Echo: An approach to enhance use case quality exploiting large language models. In *2023 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 53–60.
[18] Gabriele De Vito, Luigi Libero Lucio Starace, Sergio Di Martino, Filomena Ferrucci, and Fabio Palomba. 2025. Replication Package for the paper titled "Beyond Few-Shot Learning: Advancing LLM-Based Issue Report Classification through Intent Extraction, Ensemble Learning, and Summarization". Available at the private link: https://figshare.com/s/ebef53fca65adcedfd24. Public DOI will be made available upon acceptance.
[19] Jacob Devlin et al. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proc. naacL-HLT*. 4171–4186.
[20] Joseph L Fleiss, Bruce Levin, and Myunghee Cho Paik. 2013. *Statistical methods for rates and proportions*. john wiley & sons.
[21] Kim Herzig et al. 2013. It's not a bug, it's a feature: how misclassification impacts bug prediction. In *2013 35th intern. conf. on software engineering*. IEEE, 392–401.
[22] Maliheh Izadi et al. 2022. Predicting the objective and priority of issue reports in software repositories. *Empirical Software Engineering* 27, 2 (2022), 50.
[23] Fred Jelinek, Robert L Mercer, Lalit R Bahl, and James K Baker. 1977. Perplexity—a measure of the difficulty of speech recognition tasks. *The Journal of the Acoustical Society of America* 62, S1 (1977), S63–S63.
[24] Rafael Kallis et al. 2019. Ticket Tagger: Machine Learning Driven Issue Classification. In *Proc. of the IEEE Int. Conf. on Software Maintenance and Evolution (ICSME)*. IEEE, 406–409.
[25] Rafael Kallis et al. 2021. Predicting issue types on GitHub. *Science of Computer Programming* 205 (2021), 102598.
[26] Rafael Kallis et al. 2023. The NLBSE'23 Tool Competition. In *Proceedings of The 2nd Intern. Workshop on Natural Language-based Software Engineering (NLBSE'23)*.
[27] Rafael Kallis, Giuseppe Colavito, Ali Al-Kaswan, Luca Pascarella, Oscar Chaparro, and Pooja Rani. 2024. The NLBSE'24 Tool Competition. In *Proceedings of the Third ACM/IEEE International Workshop on NL-based Software Engineering*. 33–40.
[28] Enkelejda Kasneci, Kathrin Sessler, et al. 2023. ChatGPT for good? On opportunities and challenges of large language models for education. *Learning and Individual Differences* 103 (2023), 102274.
[29] JR Landis. 1977. The Measurement of Observer Agreement for Categorical Data. *Biometrics* (1977).
[30] Zhuoyan Li, Hangxiao Zhu, Zhuoran Lu, and Ming Yin. 2023. Synthetic Data Generation with Large Language Models for Text Classification: Potential and Limitations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 10443–10461. doi:10.18653/v1/2023.emnlp-main.647
[31] Microsoft Corporation. 2025. Azure OpenAI Service Pricing. https://azure.microsoft.com/en-us/pricing/details/cognitive-services/openai-service/#pricing Accessed: 2025-07-30.
[32] Frank Nielsen and Frank Nielsen. 2016. The MapReduce Paradigm. *Introduction to HPC with MPI for Data Science* (2016), 147–160.
[33] OpenAI. 2023. GPT-4 Technical Report. *arXiv:2303.08774* (2023).
[34] Shuyin Ouyang, Jie M Zhang, Mark Harman, and Meng Wang. 2023. An Empirical Study of the Non-determinism of ChatGPT in Code Generation. *ACM Transactions on Software Engineering and Methodology* (2023).
[35] Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2025. Qwen2.5 Technical Report. arXiv:2412.15115 [cs.CL] https://arxiv.org/abs/2412.15115
[36] Gokul Rejithkumar, Preethu Rose Anish, and Smita Ghaisas. 2024. Text-To-Text Generation for Issue Report Classification. In *Proceedings of the Third ACM/IEEE International Workshop on NL-based Software Engineering*. 53–56.
[37] Adel Remadi, Karim El Hage, Yasmina Hobeika, and Francesca Bugiotti. 2024. To prompt or not to prompt: Navigating the use of large language models for integrating and modeling heterogeneous data. *Data & Knowledge Engineering* 152 (2024), 102313.

[38] Teven Le Scao, Angela Fan, Christopher Akiki, et al. 2023. BLOOM: A 176B-Parameter Open-Access Multilingual Language Model. arXiv:2211.05100 [cs.CL]
[39] Mozhan Soltani, Felienne Hermans, and Thomas Bäck. 2020. The significance of bug report elements. *Empirical Software Engineering* 25, 6 (2020), 5255–5294.
[40] Antony Tang and Man F Lau. 2014. Software architecture review by association. *Journal of systems and software* 88 (2014), 87–101.
[41] H. Touvron, L. Martin, K. Stone, et al. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288 [cs.CL]
[42] A. Vaswani, N. Shazeer, N. Parmar, et al. 2017. Attention is all you need.. In *Advances in neural information processing systems*, Vol. 30.
[43] Chonghua Wang, Haodong Duan, Songyang Zhang, Dahua Lin, and Kai Chen. 2024. Ada-LEval: Evaluating long-context LLMs with length-adaptable benchmarks. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. 3712–3724.
[44] Jules White, Quchen Fu, et al. 2023. A prompt pattern catalog to enhance prompt engineering with chatgpt. *arXiv preprint arXiv:2302.11382* (2023).
[45] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, Anders Wesslén, et al. 2012. *Experimentation in software engineering*. Vol. 236. Springer.
[46] Fiorella Zampetti, Ritu Kapur, Massimiliano Di Penta, and Sebastiano Panichella. 2022. An empirical characterization of software bugs in open-source cyber–physical systems. *Journal of Systems and Software* 192 (2022), 111425.
[47] Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, and et al. 2022. GLM-130B: An Open Bilingual Pre-trained Model. arXiv:2210.02414 [cs.CL]
[48] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, and et al. 2023. A Survey of Large Language Models. *arXiv:2303.18223* (2023).
[49] Yu Zhou et al. 2016. Combining text mining and data mining for bug report classification. *Journal of Software: Evolution and Process* 28, 3 (2016), 150–176.