Accepted for publications in
The Journal of Systems and Software
Vol. 237, July 2026, https://doi.org/10.1016/j.jss.2026.112851

Highlights

Issue Classification with LLMs: an Empirical Study of the NASA Flight Software Systems

Giuseppe Colavito, Filippo Lanubile, Nicole Novielli, Christopher Arreza, Ying Shi

- This study compares different approaches for automatically classifying software issue reports into "bug" vs "non-bug" categories, evaluating both traditional encoder-only models (like BERT) and newer generative LLMs on NASA's Core Flight System (cFS) and F´ datasets.
- Fine-tuned encoder-only models (specifically SetFit) consistently outperformed zero-shot generative LLMs, achieving better F1 scores and bug recall. Notably, SetFit required only 40 labeled examples per class to outperform zero-shot LLMs, demonstrating that effective classification can be achieved with small, well-curated training sets rather than large labeled datasets. However, generative LLMs demonstrated comparable zero-shot performance, thus representing a viable solution in the absence of training data.
- Model performance varied considerably between the two NASA datasets. Models performed better on the F´ dataset (from a single project with immediate, author-driven labeling) compared to cFS (multiple projects with delayed, maintainer-driven labeling), highlighting how dataset characteristics and labeling practices impact classification accuracy.
- Substantially different computational costs are associated with the use of encoder-only models vs. generative LLMs. BERT-like models achieve inference in seconds on a single GPU, while generative LLMs require 25+ minutes using multiple GPUs, highlighting the trade-off between classification performance and computational resources for LLM deployment in production environments.

Issue Classification with LLMs: an Empirical Study of the NASA Flight Software Systems

Giuseppe Colavito[a], Filippo Lanubile[a], Nicole Novielli[a], Christopher Arreza[b], Ying Shi[b]

[a] *University of Bari, Italy*
[b] *NASA Goddard Space Flight Center, Maryland, USA*

Abstract

NASA collects vast amounts of problem data for space projects, which includes not only descriptions of defects but also enhancements and other issue reports. The growing complexity of Flight Software has led to an increase in the volume of problem reports, presenting both opportunities and challenges in data analysis. This paper explores AI-based solutions for classifying software issue reports in NASA's spacecraft control systems. In particular, we aim to develop an accurate classifier for identifying bug tickets, building on previous research in automated issue labeling. We conduct a benchmark study for comparing various language models and provide insights on their performance and deployment costs, with the goal of improving issue classification for NASA's growing software complexity. Based on our empirical results, we provide empirically-driven guidelines on how to address the tradeoff between the need for manual labeling of training data and the computational costs associated with the on-premise deployment of LLMs that could be used in a zero-shot setting.

Keywords: Issue Labeling, LLM, Software Maintenance and Evolution

1. Introduction

Software defect data is one of the major types of information that NASA cares about. However, accessing defect data is not as straightforward as initially thought. What is typically observed directly are discrepancies or anomalies. These are documented as problem reports or issue reports. Projects are required to document all the problems/issues found through reviews, inspections, testing, and operations. Not all the problems or issues indicate

defects. These reports can also encompass enhancements, documentation updates, or testing errors. Typical data entry includes problem descriptions, date and time, corrective actions, and root causes.

NASA has collected a vast amount of such problem/issue data for space projects in various databases, e.g., Excel, Jira, and Confluence. With the trend of increasing Flight Software (FSW) complexity and the addition of more on-board autonomy and data processing, the exponential growth of FSW in terms of size is continuing. This means more reviews, inspections, and testing need to be done. Associated with this is the growing size of problem and issue reports. This expanding dataset offers opportunities for identifying patterns, improving future software development and assurance processes, and enabling early defect detection, but also presents challenges in efficient data management, analysis, and extraction of meaningful insights. In addition, there are cases where key information (e.g. root causes) is missing, descriptions written in natural language are vague and hard to interpret and issue types (bug, enhancements, documentation, etc.) are not labeled or incorrectly labeled.

To address these challenges, besides approaches like standardizing reporting formats and adding data entry quality reviews, we aim to explore the possibility of using AI for pattern recognition, developing automated data processing systems, and creating cross-project knowledge bases. To begin with, we would like to see whether all issue reports describing defects can be correctly labeled as *bugs* using BERT, its variants or generative Large Language Models (LLMs).

More specifically, in this paper, we study the case of issue classification for the Core Flight System[1] (cFS), developed by NASA, a reusable software framework for spacecraft control systems. cFS offers modular applications, portability, and supporting tools, and has been widely adopted across NASA centers and international space agencies. The cFS architecture has proven effective in accelerating flight software deployment, promoting reuse, and establishing common standards. It is open-source and hosted on GitHub, allowing for community contributions and issue reporting. Along the same line, the F Prime[2] (F´) framework was created at Jet Propulsion Laboratory (JPL) to support the rapid development of spaceflight and embedded

[1] `https://github.com/nasa/cFS`

[2] `https://github.com/nasa/fprime`

software applications. In the context of the development and maintenance of these projects, timely and effective identification of bug reports becomes critical and their correct classification is what NASA aimed at prioritizing.

Building upon the insights from previous research [19, 5, 17, 30, 67, 15], the current study investigates the problem of automated classification of issue reports in the NASA flight systems, with the final goal of developing a reliable classifier that can accurately predict ticket labels, with a particular focus on *maximizing the recall for bug tickets*. To this aim, we design and execute a benchmark study including the cFS and F´ software projects. Our benchmark includes six generative decoder-only LLMs, two BERT-like encoder-only models, and GPT-4o.

Specifically, we make the following main contributions:

- We evaluate the performance of pre-trained language models on the bug identification task using two industrial dataset of issues from the NASA FSW system. We evaluate the models on two issue datasets by also including consideration of costs associated with the deployment of LLMs in terms of computational resources required to perform the classification task. By doing so, we aim at assessing the practical relevance and applicability in an industrial context of state-of-the-art approaches [19];

- We provide data-driven, actionable insights to improve issue classification and to inform future research work. Specifically, the results of this study reveal that the dataset characteristics and labeling practices may influence the classification performance, with more consistent issue reporting practices (as seen in the single-project F´ dataset) yielding better results than diverse but potentially inconsistent data (from the multi-project cFS dataset). This insight is particularly valuable for industrial settings where standardization of reporting practices can directly impact classification accuracy;

- We demonstrate that effective classification can be achieved with relatively small but well-curated training sets, showing that fine-tuning SetFit with a limited amount of labeled issues can outperform zero-shot LLMs. In particular, we investigate the problem of identifying the amount of training data needed to reliably fine-tune encoder-only LLMs for issue classification;

- We distribute a replication package with the scripts we use for LLM-based classification of issues, which are publicly available for reuse [3].

The remainder of this paper is structured as follows. In Section 2 we discuss the background on large language models and we provide an overview of relevant related work, by also highlighting the novel contributions with respect to previous work this study builds upon. Then we describe the two publicily available datasets we use in the current study in Section 3 and the methodology in Section 4. The empirical results of our benchmark are reported in Section 5. In Section 6, we discuss our findings, the lessons learned and the limitations of this study. Finally, we conclude in Section 7.

2. Background and Related Work

2.1. Large Language Models

Recent advancements in hardware capabilities, particularly the accessibility of GPUs, have led to significant progress in the field of NLP through the development of LLMs based on the transformer architecture [54]. These models, which leverage the concept of self-attention [54], have become the state-of-the-art for many NLP tasks [42, 23, 65]. The key to their success lies in the pre-training phase, which allows the model to learn language structure from large corpora of unlabeled data [23, 44].

Based on the transformer architecture they implement, three main types of LLMs emerged, each with distinct characteristics and applications. Encoder-only models, such as BERT [23] and its variants like RoBERTa [67], ALBERT [35], DistilBERT [47], and DeBERTa [27], are designed to use training objectives, including masked language modeling and next-sentence prediction. These models have proven effective in various natural language understanding. Another category comprises encoder-decoder models, with T5 [45] and BART [36] among the notable examples. These architectures combine the strengths of both encoding and decoding mechanisms, enabling them to handle a wide range of language tasks effectively. The third major category consists of decoder-only or generative models. This group includes prominent examples such as the GPT family [9, 41, 42], LLaMA [51, 39], Claude [3], Gemini [50], Qwen [6], Mistral [31], and Zephyr [52]. These models have

[3] `https://github.com/peppocola/IssueClassificationOnNasaFlightSoftware`

demonstrated remarkable capabilities across diverse NLP tasks, exhibiting emergent abilities that arise at large scales [58].

Each of these model types contributes uniquely to the advancement of NLP, offering different strengths and capabilities that researchers and practitioners can leverage for various applications in language technology, including Software Engineering research and practice. The application of LLMs in Software Engineering (SE) has gained significant traction in recent years, particularly with the shift towards decoder-only models like GPT-3 [9] and GPT-3.5 [41]. These models have shown remarkable abilities in several tasks including, among others, code generation and completion [63, 62], code comprehension [64], bug localization and repair [11, 12, 29, 34, 49], software testing [56], code refactoring [13, 10] and translation [38, 61]. The impact of these models on developer productivity has been notable, with reports suggesting that tools like GitHub Copilot contribute to a significant portion of code written by developers [26, 24].

2.2. Automated Issues Report Classification

Issue tracking is a widely adopted practice in software development, allowing contributors to report bugs, request features, and ask questions about software products. However, the large volume of issues submitted daily can complicate the work of developers and maintainers [8, 43, 25]. Effective issue labeling is crucial for project management and prioritization [20, 37]. However, manual labeling is time-consuming, and labels are often assigned incorrectly or not at all by contributors [28, 8, 32]. To address this challenge, researchers have proposed various approaches for automatic issue classification. Early studies used supervised learning techniques to distinguish between different types of issues [4, 28, 66]. More recently, researchers investigated the effectiveness of BERT-like models for issue report classification [30, 16, 57, 1], representing the current state-of-the-art.

Studies in this field have shown that crowd-sourced labels can be noisy and inconsistent, resulting in poor performance when training and testing models on datasets with noisy labels, which may not generalize well to real-world data [28, 16, 15, 18]. This threat makes it essential to ensure the quality of labels used for training and evaluation. In this direction, Colavito et al. [15] proposed an approach based on few-shot learning and a small, manually labeled dataset to improve the performance of issue report classification models. With the advent of generative AI, researchers have begun

investigating the potential of decoder-only LLMs for automating issue labeling [19, 5]. Empirical findings suggest that GPT-like models can represent a viable alternative to BERT-like models, especially in low-resource settings where labeled data is scarce or expensive to obtain. These models can leverage general knowledge and label explanations to classify samples, without the need for labeled data [19]. However, their performance varies significantly across datasets, and they require substantial computational resources for deployment. In contrast, BERT-like models show more consistent performance and lower resource requirements [17].

2.3. Reproducibility Challenges in LLM Research

Recent studies have highlighted significant reproducibility challenges in LLM-based software engineering research [2, 7]. Angermeir et al. [2] examined reproducibility across LLM-centric studies published at ICSE and ASE 2024, identifying critical barriers to reproduction. Of 85 articles analyzed, only 18 provided research artifacts, and merely 5 were executable. Notably, none of the five studies could be fully reproduced, with only two showing partial reproducibility.

The primary factors impeding reproducibility include: (1) incomplete or missing research artifacts, (2) unspecified dependency versions leading to API incompatibilities, (3) deprecated model access for commercial LLMs, and (4) general code quality issues. Beyond artifact-related challenges, the inherent non-determinism of LLMs poses fundamental reproducibility concerns. Even with fixed random seeds and temperature settings, LLM outputs can vary across different hardware configurations, software environments, and time periods [2].

Baltes et al. [7] developed comprehensive guidelines for empirical studies involving LLMs, emphasizing eight core principles including explicit LLM usage declaration, detailed model version and configuration reporting, complete prompt disclosure, and transparent limitation discussion. These guidelines recognize that while exact reproducibility of LLM-based experiments is challenging due to non-determinism and model evolution, transparency in reporting remains essential for scientific rigor.

To address these concerns, we provide a comprehensive replication package with pinned dependencies, containerized environments, and detailed hardware specifications (see Section 4.6). We acknowledge that despite these measures, minor variations in results may occur due to factors beyond our

control, such as floating-point arithmetic precision and inference implementation details across different systems [2, 7].

2.4. Novel Contributions

In this study, we aim to further the understanding of the potential of LLMs in the issue classification scenario, specifically in a real-world use case where the timely identification and management of bug reports are paramount. To this end, we evaluate a series of classifiers based on both BERT-like models and generative models based on decoder-only transformers. Beyond assessing the models' performance, we aim to provide an understanding of the feasibility of their adoption in the use-case scenario of the NASA flight software concept as well as in terms of computational costs associated with LLM deployment and usage.

This study builds upon and extends the benchmark study by Colavito et al. [19] by providing industrial validation in high-stakes aerospace applications where software reliability directly impacts safety and mission success. Unlike previous research that focused primarily on multiple GitHub projects, we examine dedicated NASA flight software datasets with consistent reporting practices and domain-specific requirements. Furthermore, While previous studies provided limited analysis of practical deployment considerations, our research addresses the critical gap between academic findings and industrial implementation by examining computational costs, hardware requirements, privacy concerns, and resource constraints essential for production deployment in sensitive environments, following recent benchmarking approaches [17]. Through comprehensive evaluation of both BERT-like encoder-only models and generative decoder-only LLMs, systematic analysis of data efficiency requirements, and establishment of cost-benefit frameworks, this study provides the practical insights and validation needed for organizations to confidently implement LLM-based issue classification in safety-critical domains where incorrect classification can have mission-threatening consequences.

3. Dataset

3.1. The NASA Flight Software Projects

The FSW system, which acts as the spacecraft's brain, is the central control system of a spacecraft that manages all critical functions, ensures safety, and provides autonomous operation with guaranteed performance and reliability. The cFS, developed by NASA Goddard Space Flight Center (GSFC), is a platform and project independent reusable software framework and set

of reusable software applications that provides 1) a dozen modular software applications for common functions, e.g. limit checker, file transfer, memory management, etc. 2) portability via processor and OS abstraction; 3) supporting tools (unit test framework, performance, ground system simulator, etc.).

The cFS architecture has been used at six NASA centers by 40+ projects historically including Roman Space Telescope, Capture, Containment and Return System, Lunar Reconnaissance Orbiter and Artemis. cFS has been proven to accelerate the deployment of quality flight software, reduce project uncertainties, promote reuse and collaboration, simplify maintenance, support innovation, and establish common standards across NASA missions. cFS is now a standard for the space community as the standard FSW framework (for NASA, ESA, CSA, JAXA) and is adopted by DoD and other US agencies, academia, and private sectors. The cFS project is open source and hosted on GitHub, where users can report issues, request new features, and contribute to the development of the software. As a first dataset, we collected a dataset of cFS issue reports from the cFS GitHub repository and its submodules. The dataset consists of 2,724 issue reports, of which 827 are labeled as bugs.

F´ is a newer, component-based framework designed for the swift development and deployment of spaceflight and other embedded software applications. Initially created at the JPL, F´ has been effectively utilized in various JPL missions like Mars Helicopter Ingenuity, Europa Clipper, and Psyche. It is specifically optimized for, but not restricted to, small-scale spaceflight systems like CubeSats, SmallSats, and scientific instruments. The second dataset consists of issue reports from the F´ GitHub repository. The dataset includes 751 issue reports, of which 371 are labeled as bugs.

In Table 1, we report the distribution of the labels for both datasets.

	Projects			
Label	**cFS**		**F´**	
Bug	827	(30%)	371	(50%)
Non-Bug	1897	(70%)	380	(50%)

Table 1: Distribution of issue labels in the two datasets used for this study, which include issue reports from the NASA cFS and F´ projects.

3.2. Data Quality Assessment

In both datasets, a small percentage of the issues (3% for cFS, 5% for F´) had multiple labels assigned. To reduce the noise in the data, thus enhancing the datasets' quality, we conducted a manual annotation of such issues, to disambiguate their labels. Specifically, we extracted the issues with multiple labels, i.e. 84 issues from cFS and 40 from F´, totaling 124 cases. Two independent annotators manually labeled these 124 issues. To avoid any biases in the annotation, they could not access the existing labels. Each annotator could assign a single label to each issue (either *bug* or *non-bug*), basing their decision solely on the issue report's title and textual content. This approach enabled the reduction of bias and ensured a more accurate labeling of the tickets.

To evaluate the reliability of the annotated process and, therefore, of the resulting gold standard dataset, we calculated Cohen's kappa as a measure of the agreement between the annotators. For the cFS dataset, we observed that Cohen's kappa [14] was 0.80, indicating substantial agreement. For the F´ dataset, Cohen's kappa was 0.90, indicating almost perfect agreement [55]. After the round of individual labeling, the two annotators discussed and solved the disagreement cases (12 issues). For 4 issues, the annotators were not able to reach a consensus due to a lack of contextual information and decided to discard them. As a result, the final datasets include 2,724 (cFS) and 751 (F´), respectively, as reported in Table 1.

4. Methodology

4.1. Choice of Models

The choice of the models to include in our benchmark grounds on evidence provided by existing research on this topic. Several ML-based models have been proposed for issue report classification. State-of-the-art models are based on pre-trained language models such as BERT [23], RoBERTa [67], SetFit [53]. More recently, LLMs [9, 33] have been proposed, especially using zero-shot learning for low-resource settings.

In this study, we consider *six generative models* of different sizes, built by well-known organizations, which we selected based on their popularity and availability for public reuse on the Hugging Face model hub. We group the generative models based on the hardware required for their deployment on-premise. The first group includes models that fit into 2 Nvidia A100 64GB GPUs, while the second group includes models that fit into 4 Nvidia A100

64GB GPUs (see Table 2). To reduce the memory footprint and speed up the inference time, all models are 4-bit quantized before being prompted, using the *bitsandbytes* library [21].

	Hardware Used	
	2xA100 64GB	
Organization	**Model Name**	**Parameters**
meta-llama	Meta-Llama-3.1-8B-Instruct	8b
mistralai	Mistral-7B-Instruct-v0.3	7b
Qwen	Qwen2.5-7B-Instruct	7b
	4xA100 64GB	
meta-llama	Meta-Llama-3.1-70B-Instruct	70b
mistralai	Mixtral-8x7B-Instruct-v0.1	46b
Qwen	Qwen2.5-72B-Instruct	72b

Table 2: Generative models included in the benchmark grouped based on the hardware requirements for deployment on premise.

Furthermore, we also include *two encoder models*, namely RoBERTa and SetFit. This choice is motivated by empirical findings reported in previous research reporting state-of-the-art performance for these two BERT-like models for the task of automated issue labeling [15]. In particular, RoBERTa is included as a classifier based on this BERT variant achieved state-of-the-art performance in previous benchmark studies on issue labeling [16, 15]. Furthermore, we include consideration of few-shot learning techniques to issue classification leveraging the SetFit framework [53]. SetFit is optimized for fine-tuning transformer models such as Sentence-BERT (SBERT) [46] on limited training data. Sentence transformers are a family of models that use transformer architectures to generate sentence embeddings. This approach is particularly effective for semantic similarity tasks, for which BERT-like models are not optimized [46]. While not outperforming the RoBERTa baseline, SetFit achieves comparable performance when trained and tested on data for which label consistency was manually verified, as demonstrated in previous work [15].

Finally, we include consideration of models that can be accessed through the *OpenAI API*, for completeness. Specifically, we consider the latest GPT

model at the time of writing, i.e. GPT-4o-2024-05-13 [42]. We include GPT-4o based on the empirical evidence provided by previous investigation of Colavito et al. [19] on the use of GPT-like models for automated issue labeling. Specifically, they found that they achieve a performance comparable to state-of-the-art BERT-like LLMs [15].

However, being proprietary, GPT-like models introduce external dependencies and may trigger privacy-related concerns in case of issues including sensitive data. In such cases, on-premise generative LLMs might represent an alternative solution, albeit posing deployment challenges due to computational and scalability issues. As such, we compare their performance and inference time with two state-of-the-art BERT-like encoder-only models.

4.2. Preprocessing

The issue text was kept as is. For BERT-like models, the title and body of the issue were concatenated, while for generative models they were kept separate and used in the prompt.

4.3. Fine-tuning Encoder Models

To evaluate the performance of the SETIFT and RoBERTa models, we replicate the experimental setting of previous empirical studies [15, 17]. As for SetFit we replicate its fine-tuning and evaluation approaches by executing the scripts from the replication package distributed with the original study [4].

The dataset was split into train and test sets using an 80/20 split ratio. The split was performed using the `scikit-learn` library, using stratification in order to keep the distribution of labels in the training and test sets consistent with the original dataset. For both datasets, we use the train set to fine-tune the SetFit and RoBERTa models; then, we test the performance of all models, including the generative ones, on the test set. RoBERTa is trained for 15 epochs using a learning rate of $2e-5$, a weight decay of $1e-2$. The base sentence-transformers model used for SetFit is `sentence-transformers/all-mpnet-base-v2`. It is trained for a single epoch and with 20 iterations [5]. For both models, the batch size is set to 16. For the main evaluation reported in Table 4, we report results from a single training run using a fixed random seed (seed=42) and a deterministic

[4] `https://github.com/collab-uniba/Few-Shot-Learning-for-Issue-Report-Classification`

[5] `https://huggingface.co/docs/setfit/conceptual_guides/sampling_strategies`

stratified train-test split. We note that the learning curve analysis presented in Section 5.2, in which we trained 10 SetFit models per sample size using different random seeds and averaged the results, was specifically designed to account for variance due to random initialization.

Table 3 reports the number of samples in the training and test sets of cFS and F´ datasets.

Label	**Train**		**Test**		**Train**		**Test**	
Bug	662	(30%)	165	(30%)	296	(49%)	75	(50%)
Non-Bug	1517	(70%)	380	(70%)	304	(51%)	76	(50%)
Total	2179		545		600		151	
	cFS Dataset				F´ Dataset			

Table 3: Distribution of labels in the cF and F´ datasets and their train-test split. For each dataset, the train set is used for fine-tuning BERT-like models while the test set is used to evaluate the performance of all models, including generative LLMs.

4.4. Prompting Generative Models

The prompt template is designed to guide the model in classifying issue reports by providing clear instructions and label definitions. It includes a system message that specifies the model's role, followed by a user message detailing the format and task requirements. The template is structured to elicit a step-by-step reasoning process from the model before assigning a label. This approach aims to leverage chain-of-thought reasoning, which has been shown to enhance performance in GPT-like models [59]. The prompt was adapted from the one designed and validated in previous studies by Colavito et al. [19, 17]. The main difference is that we included a system message that defines the model's role as an assistant specialized in analyzing GitHub issues and assigning appropriate labels. The prompt structure and its components are represented in Figure 1.

The system is designed as an assistant specializing in analyzing GitHub issues and assigning appropriate labels, as specified in the *system message*. This setup provides context for the task the model is expected to perform. The *input format* outlines the structure of the issue report, which includes the title and body of the issue. This ensures that the necessary information is presented in a standardized way for effective analysis. The task involves

GitHub Issue Classification Prompt

System Message:
You are an AI assistant specialized in analyzing GitHub issues and assigning appropriate labels.
Input Format:
You are provided with a GitHub issue in this format:
Title: """the title of the issue"""
Body: """the body of the issue"""
Task Description:
You have to assign it a label.
Possible labels are: "bug", "non-bug"
Label Descriptions:
The "bug" label is used to identify an issue report that describes a problem or error within the software or codebase. It indicates that something is not functioning as intended or producing unexpected results. Bug reports help developers identify and fix issues to improve the overall quality and reliability of the software.
The "non-bug" label is applied to any issue that is not a bug. This includes feature requests, questions, documentation improvements, or any other type of issue that does not describe a malfunction in the software.
Input Issue:
Title: """title"""
Body: """body"""
Output Format Instructions:
The output should be a markdown code snippet formatted in the following schema, including the leading and trailing "“‘json" and "“‘":

```
```json
{
"reasoning": "string",  // The step-by-step reasoning to
   assign the correct label to the issue.
"label": "string"  // The label to assign to the issue.
   Possible labels are: "bug", "non-bug".
}
```
```

Output:

Figure 1: GitHub Issue Classification Prompt

classifying the issue and assigning it a label and is explained in the *task description*. To assist in this, a list of possible labels is provided along with detailed *label descriptions*. These descriptions are intended to guide the model in making accurate and consistent classifications. When a specific issue is presented for classification, it is called the *input issue*. The model's output must follow a specific format, i.e., a JSON object with two fields, as defined in the *output format instructions*. Specifically, the first field, "reasoning," is a string that contains a step-by-step explanation of the thought process behind the assigned label. The second field, "label," is a string representing the assigned label, which must be either "bug" or "non-bug."

We use a zero-shot approach where the prompt only contains label descriptions without any labeled examples. This decision is based on findings from previous work [19], which indicated no performance gain in a few-shot learning setting, compared to a zero-shot setting.

For on-premise models, we employed greedy decoding, which guarantees deterministic outputs given fixed model parameters and execution conditions. For proprietary models (e.g., GPT-based APIs), we set the temperature to 0 to reduce output variability; however, this does not strictly ensure determinism, as such systems may still exhibit non-deterministic behavior due to implementation-specific factors

4.5. Parsing the output of generative models

The prompt is designed to require generative models to output a JSON object that includes both the label and the reasoning behind it. However, inconsistencies in output formatting are common, such as missing commas or brackets, which complicate the parsing process. Occasionally, the output may consist of multiple JSON objects or a JSON object followed by a string. To address these issues, we utilize a regular expression-based extraction strategy. We search for the first occurrence of the pattern `"label": ".*"` to extract the label. Outputs not matching this pattern are deemed invalid. This approach to handling inconsistent output formats is informed by recent studies on issue labeling [17] and on code translation tasks [38], which emphasize the need for specific extraction methods to ensure accurate model evaluation and benchmarking.

4.6. Evaluation

We evaluate the classification performance of all models included in the benchmark in terms of Precision, Recall, and F1, in line with consolidated

practice. We provide the F1 score for each label, as well as the overall micro- and macro-averaged F1 scores since micro-averaging is known to be influenced by the performance of the majority class [48]. Furthermore, we provide the inference time for all models. Specifically, we report the time that each model takes to classify the entire test set to enable a full assessment of the feasibility of LLM-based classification in practice.

4.7. Reproducibility Measures

To facilitate reproduction of our results and address known challenges in LLM reproducibility [2, 7], we provide a comprehensive replication package[6] with the following components:

Dependency Management. We provide a `requirements.txt` file with fully pinned versions of all dependencies to prevent API drift and version mismatches that commonly impede reproduction [2].

Containerization. We include a Dockerfile that creates a fully reproducible containerized environment with exact library versions and system dependencies, ensuring platform independence and minimizing environment-related variability.

Hardware and System Specifications. All experiments were conducted on NVIDIA A100 64GB GPUs, with 128GB RAM, running Ubuntu 22.04 LTS. We report this information as hardware configurations can significantly impact inference behavior and performance [2].

Experimental Configuration. We used random seed 42 throughout all experiments to enhance reproducibility. All configuration files correspond exactly to the experiments reported in this paper and are included in the replication package with clear documentation.

Verification Tools. We provide an end-to-end smoke test script that verifies correct environment setup and produces expected output formats. Additionally, we include step-by-step instructions for exact reproduction, including expected outputs and troubleshooting guidance.

Despite these measures, we acknowledge that LLM-based experiments face inherent reproducibility limitations. Even with temperature=0 and fixed seeds, minor variations may occur due to floating-point precision, hardware-specific optimizations, and non-deterministic operations in deep learning

[6] `https://github.com/peppocola/IssueClassificationOnNasaFlightSoftware`

frameworks [2]. We expect these variations to be negligible and not affect the overall conclusions of our study.

5. Results

5.1. Assessing the performance of LLMs on the NASA datasets

In the following, we report the results of the issue classification on the two datasets. In Table 4, we compare encoder models for the cFS and F´ datasets. In Table 5, we compare the performances of the generative models on the cFS and F´ datasets. In Table 6, we report the performance of GPT-4o. Note that GPT can only be used via an API call, and as such, inference time is not comparable to the other generative models.

Table 4: Comparison of RoBERTa and SetFit performances on cFS and F´ Datasets. The best classification performance for each dataset is highlighted in bold.

	cFS Dataset						**F´ Dataset**					
	RoBERTa			SetFit			RoBERTa			SetFit		
Label	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Bug	.71	.70	.71	**.77**	**.83**	**.80**	.90	.96	.93	**.95**	**.96**	**.95**
Non-Bug	.87	.88	.87	**.92**	**.89**	**.91**	**.96**	.89	.93	**.96**	**.95**	**.95**
Micro Avg.	.82	.82	.82	**.88**	**.88**	**.88**	.93	.93	.93	**.95**	**.95**	**.95**
Macro Avg.	.79	.79	.79	**.85**	**.86**	**.86**	.93	.93	.93	**.95**	**.95**	**.95**
Training Time	12m 27s			41m 46s			03m 46s			11m 34s		
Inference Time	00m 06s			00m 02s			00m 01s			<00m 01s		
GPU	1xA100-64GB											

For both datasets, we observe that SetFit outperforms RoBERTa (see Table 4) as well as all generative models (see Tables 5 and 6) in terms of overall F1 score (both micro- and macro-average). It is worth noting that generative models were prompted with zero-shot examples (hence the absence of the estimated training time in Tables 5b and 6). This makes them still a valuable choice in case of the absence of labeled data that could be used for training. We also note that the inference time for BERT-like models is significantly lower (up to 6" on 1 GPU) than the time required for classifying the test set using generative LLMs (25' for the best-performing model LLama-3.1, which requires 4 GPUs for deployment).

The results also show a marked difference in performance between the cFS and F´ datasets with all models. In search of an explanation, we analyzed

Table 5: Comparison of Performances for Generative Models on the cFS and F´ Datasets. The models are grouped according to the number of GPUs required for their deployment, with the best classification performances highlighted in bold within each group.

(a) cFS Dataset

	Qwen2.5-7B			Mistral-7B-v0.3			Llama-3.1-8B			Qwen2.5-72B			Mixtral-8x7B			Llama-3.1-70B		
Label	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Bug	**.71**	.53	**.61**	.48	**.84**	**.61**	.45	**.84**	.59	.65	**.79**	**.71**	**.74**	.64	.68	.65	**.79**	**.71**
Non-Bug	.85	**.71**	**.77**	**.90**	.61	.73	.89	.56	.68	**.90**	.81	**.85**	.86	**.85**	**.85**	**.90**	.81	**.85**
Micro Avg	**.81**	.66	**.73**	.68	**.68**	.68	.64	.64	.64	.81	**.81**	**.81**	**.83**	.79	**.81**	.81	**.81**	**.81**
Macro Avg	**.78**	.62	**.69**	.69	.73	.67	.67	**.70**	.64	.77	**.80**	**.78**	**.80**	.74	.77	.77	**.80**	**.78**
Inference Time	01h 11m 58s			00h 44m 28s			00h 33m 43s			01h 39m 09s			01h 49m 29s			01h 33m 51s		
GPU	2xA100-64GB									4xA100-64GB								

(b) F´ Dataset

	Qwen2.5-7B			Mistral-7B-v0.3			Llama-3.1-8B			Qwen2.5-72B			Mixtral-8x7B			Llama-3.1-70B		
Label	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Bug	**1.00**	.57	.73	.90	.88	**.89**	.85	**.89**	.87	**.97**	.85	**.91**	.96	.63	.76	.94	**.87**	.90
Non-Bug	.82	.80	.81	.88	**.91**	**.90**	**.89**	.84	.86	.87	**.97**	**.92**	.73	.97	.83	**.88**	.95	.91
Micro Avg	**.89**	.69	.78	**.89**	**.89**	**.89**	.87	.87	.87	**.91**	**.91**	**.91**	.80	.80	.80	**.91**	**.91**	**.91**
Macro Avg	**.91**	.69	.77	.89	**.89**	**.89**	.87	.87	.87	**.92**	**.91**	**.91**	.84	.80	.79	.91	**.91**	**.91**
Inference Time	00h 09m 12s			00h 11m 20s			00h 20m 14s			00h 28m 02s			00h 31m 33s			00h 25m 09s		
GPU	2xA100-64GB									4xA100-64GB								

Table 6: GPT Performance on the cFS and F´ Datasets.

Dataset	**cFS**			**F´**		
Label	P	R	F1	P	R	F1
Bug	.67	.80	.73	.96	.88	.92
Non-Bug	.90	.83	.86	.89	.96	.92
Micro Avg	.82	.82	.82	.92	.92	.92
Macro Avg	.79	.81	.80	.92	.92	.92
Inference Time	14m 02s			03m 22s		

the temporal characteristics of issue labeling practices in both datasets. Our analysis reveals significant differences in the labeling workflows implemented in the two projects. In the F´ dataset, 91% of issues receive labels at the time of creation, with non-instantly labeled issues having an average delay of 170 hours. In contrast, only 41% of cFS issues are labeled at the creation time, with the remaining issues experiencing an average labeling delay of 999 hours. Additionally, 98% of F´ issues are labeled by their original authors, compared to 70% in cFS. These patterns suggest that F´ follows a more immediate and author-driven labeling process, while cFS relies more heavily on post-hoc labeling by maintainers or other contributors. Therefore, the F´ dataset benefits from immediate labeling and consistent author-driven classification, resulting in more coherent and contextually accurate labels. The substantially longer labeling delays in cFS introduce temporal disconnection between issue creation and classification, potentially leading to label noise as contributors may forget context or misinterpret issues over time. Furthermore, the F´ dataset is derived from a single project, which results in a more homogeneous approach to issue creation and labeling by the project contributors and users. Conversely, the cFS dataset encompasses issues from multiple projects. This difference can be the cause of discrepancies in the labeling of the issues, which can affect the performance of the models, as also suggested by previous findings [16, 18].

5.2. Minimum train set size to build a robust issue label classifier

Minimizing the amount of labeled data needed to train a model is crucial in practice, as labeling data may not be available or may be expensive to obtain. The results of the classification study suggest that the encoder-

only models outperform classifiers based on generative models consistently on both datasets. However, the manual annotation of training data remains a time-consuming task, requiring a significant amount of hours for labeling and data quality assurance, even for a small, curated dataset. Furthermore, new projects have few, if any, issues that can be made available for annotation, thus posing the question on how to transfer the current findings to future projects.

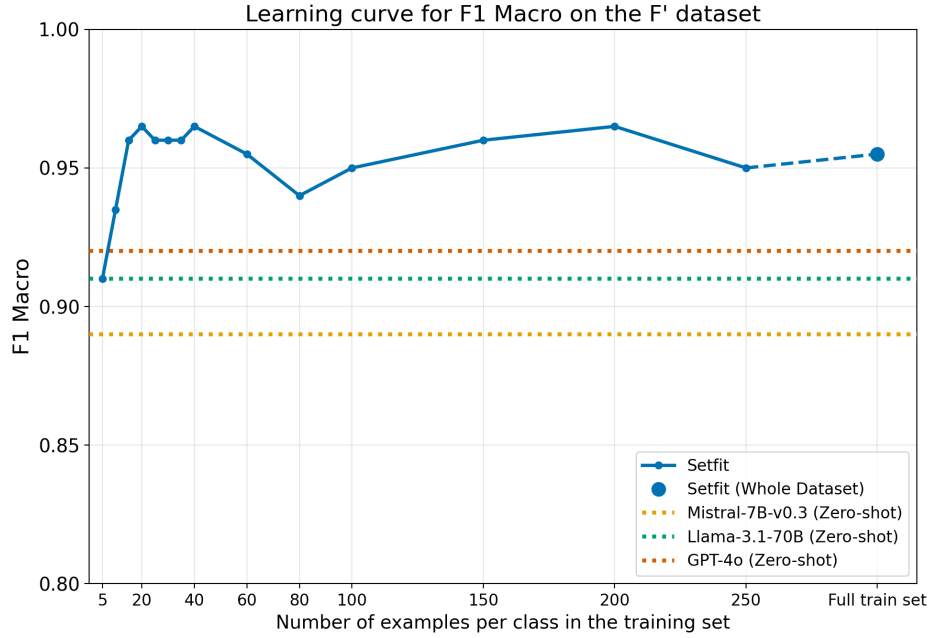
As such, we perform a follow-up study to establish the optimal number of labeled examples needed to train the models and achieve good performance, in line with what is reported in Table 4. This was done by plotting and analyzing the learning curves of the best-performing model (SetFit) for the F´ dataset. This was done by progressively increasing the number of training examples and comparing their performance to zero-shot LLMs.

We started by randomly sampling 5 examples per class from a balanced dataset. For each sample size, we trained 10 models, each with a different random seed, to account for variance in the results. The performance metrics from the 10 models were then averaged to obtain a robust estimate. We repeated this process with steps of 5 from 5 to 100 samples per class, and then with steps of 50 from 100 to 250. The performance of the models obtained at each iteration is consistently assessed on the full test set.

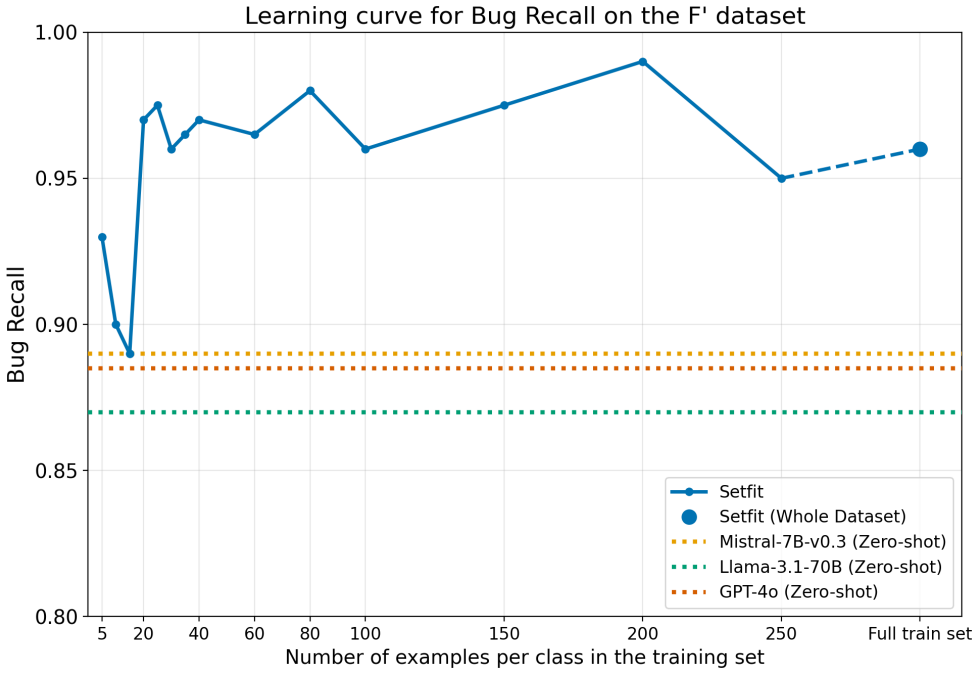
The learning curves are shown in Figures 2a, which depicts the curve for the overall F1 macro measure, and 2b, which represents the curve for the Recall of the *bug* class, for which high recall is crucial to ensure adequate and timely recognition and handling. The dotted lines represent the performance of generative LLMs used in a zero-shot setting, i.e. in the absence of training data. The learning curves suggest that SetFit outperforms classifiers based on generative LLMs already with less than 20 labeled examples, with optimal performance achieved around 40 examples in the training set in terms of both F1 and bug recall.

6. Discussion

Generative vs. encoder-only LLMs. The empirical evidence provided by this study both corroborates and expands upon the findings of previous research on GPT-like LLMs for automated issue labeling. In their earlier work Colavito et al. [19] observed that GPT-3.5, without any task-specific fine-tuning, achieved performance comparable to the fine-tuned SetFit baseline on a manually labeled dataset. This zero-shot capability of GPT-3.5 suggested



(a) F1 Macro Comparison.



(b) Bug Recall Comparison.

Figure 2: Learning curves for F1-Macro and Bug Recall on the F´ dataset.

that LLMs could effectively classify issue reports with minimal performance loss compared to fine-tuned BERT-like models, offering a significant advantage in scenarios lacking labeled data. However, the present study yields more nuanced results so we could partially confirm previous findings. Using the latest GPT-4o model, we could replicate a comparable performance in the zero-shot setting only for the F´ dataset (Overall F1 = .92, Recall for *bug* = .88), which is composed of issues from a unique software project, whereas the performance for cFS is substantially lower (Overall F1 = .80, Recall for *bug* = .80). Similar observations hold for the on-premise LLMs, for which we observe diverse performances in the zero-shot setting, with an overall F1 = .91 and Recall for *bug* = .89 with Llama-3.1-70B for the F´ dataset, which is composed of issues from a unique software project, whereas the performance for cFS is substantially lower (Overall F1 = .78, Recall for *bug* = .79 with Llama-3.1-70B).

These findings highlight the variability in generative LLMs' zero-shot performance across different datasets, in line with evidence provided by a comprehensive benchmark study assessing 22 generative decoder-only LLMs and 2 BERT-like encoder-only models for issue classification performed on two different datasets of GitHub issues [17]. While generative LLMs offer a viable solution in the absence of training data, their inconsistent performance imposes a careful preliminary validation of the classification outcome before adoption in practice, to ensure alignment with requirements for the envisaged use cases. Conversely, supervised training of smaller, encoder-only models consistently yields state-of-the-art performance, albeit at the cost of requiring labeled data (see Table 4), thus further confirming previous empirical evidence [17].

Trade-off Between Labeling and Computational Costs. Our model obtained by performing few-shot fine-tuning of SetFit achieves the best performance on both datasets. Llama-3.1-70B achieves comparable performance in a zero-shot setting suggesting that on-premise LLMs may be used in absence of training data. However, the analysis of computational costs suggests that deployment on-premise of LLMs is associated with higher inference time and hardware requirements. Conversely, the inference time for SetFit is significantly lower, i.e. up to 6" on a single GPU. This is an important aspect to consider when identifying the solution to adopt in a real use-case scenario. While zero-shot usage of on-premise LLMs might be good for rapid prototyping of classifiers in the absence of training data, better performance can be achieved by fine-tuning BERT-based models with small manually curated

training data.

It is important to highlight that for BERT-like models, fine-tuning is a necessary step, albeit a relatively inexpensive process, both in terms of computational resources (training time put to 12' in the worst case, using the entire train set) and human effort required for labeling the training data. In particular, the analysis of the learning curves suggests that fine-tuning of SetFit with 40 issues already produces higher overall F1 as well as higher recall for the *bug* class compared to all generative models included in this benchmark. In contrast, the cost associated with fine-tuning generative LLMs is known to be significantly higher in terms of both computational resources and data required [22]. As an alternative solution to reduce computational costs, GPT might be considered. However, further limitations might arise due to the introduction of unwanted external dependencies and potential privacy issues associated with the use of the OpenAI API calls.

6.1. Lessons Learned

Overall, our empirical findings suggest that successful issue classification depends more on thoughtful data curation, appropriate model selection for specific constraints, and systematic process implementation than on simply deploying the largest available models. The findings of this study demonstrate that practical, cost-effective solutions often outperform resource-intensive alternatives when properly implemented. As such, based on our findings, we provide empirically-driven guidelines for the practical adoption of LLMs for automated issue label classification in an industrial project.

Data Quality Guidelines

- *Prioritize consistency over quantity.* The study demonstrates that classifiers evaluated on well-curated, single-project datasets (like F´) significantly outperform models evaluated on larger but diverse multi-project datasets (like cFS). Developers should focus on establishing standardized issue reporting practices within their organizations rather than simply collecting more data. Consistent labeling conventions and reporting formats yield better classification results than large volumes of inconsistently labeled issues.

- *Start small but aim for quality.* The learning curve analysis reveals that effective classification can be achieved with as few as 40 carefully labeled issues per class. Rather than attempting to label thousands of

issues, developers should invest in creating a small, high-quality training set with verified labels and clear annotation guidelines.

Model Selection and Deployment

- *Choose models based on your constraints, not just performance.* While larger generative models can achieve good zero-shot performance, fine-tuned smaller models like SetFit often provide superior results with significantly lower computational requirements. For production systems with labeled data available, BERT-like models requiring only 6 seconds for inference should be preferred over LLMs requiring 25+ minutes using the same hardware. Zero-shot LLMs have proven valuable for rapid prototyping and scenarios without labeled data, but they require substantial computational resources for production use. Developers should evaluate not just model accuracy but also inference time, hardware requirements, and ongoing operational costs when making model selection decisions.

- *Consider hybrid approaches for different use cases.* Use zero-shot LLMs for initial triage or when dealing with novel issue types, but rely on fine-tuned models for production classification of well-defined categories. This allows organizations to benefit from both approaches while managing computational costs effectively.

- *Plan for your data sensitivity requirements.* Organizations handling sensitive or proprietary information should prioritize on-premise solutions over API-based services. The study shows that fine-tuned models can be deployed locally with reasonable resource requirements, while maintaining data privacy and eliminating external dependencies.

Issue Creation and Labeling

- *Implement immediate label assignment upon issue creation* The performance difference between single-project and multi-project datasets highlights the importance of consistent reporting practices, but equally critical is the timing of label assignment. Organizations should implement workflows that encourage or require issue authors to assign preliminary labels at the creation time, as immediate labeling by the original author significantly improves dataset quality and subsequent

classification accuracy. Extended delays between issue creation and labeling introduce substantial context loss, contributing to performance degradation across all model types. Having a strong set of guidelines for issue reporting, such as assigning labels at the time of issue creation, can significantly improve classification outcomes.

6.2. Limitations

In the following, we discuss the limitations of the present study and how we mitigate them. Despite our efforts to ensure reproducibility through pinned dependencies, containerization, and detailed documentation, LLM-based experiments face inherent reproducibility challenges [2, 7].

Recent empirical evidence demonstrates that even with temperature=0 and fixed random seeds, LLM outputs may exhibit minor non-determinism due to factors including floating-point operations, hardware differences (e.g., different GPU architectures), CUDA version variations, and implementation-specific optimizations in inference libraries [2]. This non-determinism affects both commercial models accessed via APIs and self-hosted open models.

For encoder-only models (RoBERTa, SetFit), we used fixed random seeds (seed=42) and deterministic train-test splits, which provide more stable reproduction conditions. However, for any model employing neural network architectures, complete bit-level reproducibility across different hardware and software environments cannot be guaranteed.

The rapid evolution of the Hugging Face ecosystem presents additional challenges. API changes and dependency updates can break compatibility even with pinned versions if underlying system libraries change [2]. To mitigate this, our replication package includes fully pinned environment specifications and containerization via Docker for environment isolation.

Furthermore, commercial model providers do not guarantee long-term availability of specific model versions. Although we specify exact model versions where possible (e.g., gpt-4-0613), these versions may be deprecated in the future, limiting long-term reproducibility of experiments using commercial APIs [2]. This motivates our inclusion of open-source baselines where feasible.

The design of the prompt template for generative LLMs introduces a potential threat to internal validity. Previous research has demonstrated the potential impact of prompt engineering in optimizing the classification performance of generative models [9]. To address this concern, we defined the prompt in line with the findings of previous research by Colavito et

al. [19] who employ multiple prompts of varying lengths, including zero-shot approaches with only label definitions and few-shot methods incorporating labeled issue examples. Specifically, they previously explored diverse strategies for prompt definition in the few-shot prompt setting, such as including or omitting label descriptions alongside issue examples. They conclude that, while including label descriptions enhances performance, few-shot prompt strategies are less effective than zero-shot approaches for generative LLMs. As such, in the current benchmark we decided to focus solely on zero-shot evaluation for generative LLMs.

Internal validity may also be affected by configuration parameters, a common limitation in ML-based approaches. For generative LLMs, the model temperature is a crucial setting, as higher values can lead to more unpredictable outputs (as documented for GPT-like models [40]). While this unpredictability can be an advantage for creative tasks, it may be detrimental for classification tasks, potentially resulting in divergent outputs for identical prompts [60]. For on-premise models, we mitigated this threat by employing greedy decoding, which guarantees deterministic outputs under fixed execution conditions. For proprietary models (e.g., GPT-based APIs), we set the temperature to zero to minimize output variability; however, this setting does not strictly guarantee determinism due to implementation-specific factors such as dynamic decoding behavior and backend-level effects. Future research could explore the impact of temperature and decoding strategies on classification tasks.

An additional limitation of this study is the fact that it does not consider the potential for improved classification performance leveraging fine-tuning of generative LLMs with software development domain data. While worth investigating, this problem is outside the scope of our current study. In fact, our primary focus is on evaluating the performance of BERT-like and generative LLMs in a specific use-case scenario, i.e. on the task of issue labeling in the NASA Flight Software Systems. Future replications incorporating fine-tuned generative models could complement our findings, providing more comprehensive empirical evidence to guide researchers and practitioners in LLM usage.

As for external validity, while our study focuses on aerospace software, several findings likely generalize to other safety-critical domains (automotive, medical devices, financial systems) that share similar characteristics: formal reporting processes, rigorous quality requirements, and specialized terminology. However, the better performance on single-project datasets, as in the

case of F´, may suggest further investigations are needed to further confirm the current findings in large software ecosystems, for which multi-project inconsistency might be a relevant issue.

A threat to construct validity concerns our operational definition of "bug." Our classification relies solely on issue report text and assigned labels, without code analysis or design artifact review. This represents a significant limitation, as accurate bug identification in practice requires examining source code, consulting requirements specifications, and obtaining expert input to distinguish genuine defects from misunderstandings, feature requests, or documentation issues. Our manual annotation achieved high inter-annotator agreement (Cohen's $\kappa = 0.80$ for cFS, $\kappa = 0.90$ for F´), demonstrating label *consistency*. However, this measures agreement between annotators rather than correctness against ground truth. Both annotators based their decisions on issue text alone, without access to code or consultation with NASA developers. Therefore, we cannot guarantee that all issues labeled as "bugs" represent true defects, nor that "non-bug" labels are definitively correct. Our work addresses *automated issue triage* based on textual content. The models learn to predict labels consistent with NASA repository practices, which may themselves contain errors. While this provides practical value for rapid triage and prioritization, practitioners should treat predictions as triage suggestions rather than definitive classifications, implement human review for high-priority issues, and validate critical decisions with code-level analysis, particularly in safety-critical aerospace applications where misclassification could have serious consequences.

7. Conclusion

This study presents a comprehensive benchmark comparing BERT-like encoder-only models and generative LLMs for issue classification in NASA Flight Software Systems. While this study builds upon state-of-the-art classification techniques [15, 19], it introduces significant contributions in terms of empirically-driven guidelines for industrial deployment of LLMs for automated issue labeling in safety-critical domains. Specifically, this study further contributes domain-specific assessment of LLMs for issue classification on two NASA datasets that are representative of unique characteristics of aerospace software development practices. These insights can guide practitioners in selecting appropriate models based on their specific needs, available resources, and data constraints.

Future research directions include exploring the prediction of more fine-grained labels within the bug class, to enable early defect detection and profiling. This could lead to a more precise categorization of software issues, potentially improving the efficiency of bug triage and resolution processes. Additionally, examining the potential for transfer learning between different software projects or domains could leverage pre-existing knowledge and improve classification accuracy. Such research could pave the way for more robust and adaptable issue classification systems, capable of performing well across varied software environments and project types.

By advancing our understanding of issue classification in flight software systems, this work contributes to the broader goal of enhancing software quality assurance practices in critical domains. The methodologies and findings presented here may also inform similar classification tasks in other specialized software development fields, potentially leading to improved software maintenance and reliability across various sectors of the industry.

Acknowledgments

This research was co-funded by the NRRP Initiative, Mission 4, Component 2, Investment 1.3 - Partnerships extended to universities, research centres, companies and research D.D. MUR n. 341 del 15.03.2022 – Next Generation EU ("FAIR - Future Artificial Intelligence Research", code PE00000013, CUP H97G22000210007) and by the European Union - NextGenerationEU through the Italian Ministry of University and Research, Projects PRIN 2022 ("QualAI: Continuous Quality Improvement of AI-based Systems", grant n. 2022B3BP5S, CUP: H53D23003510006). The models are deployed on the Leonardo supercomputer with the support of CINECA-Italian Super Computing Resource Allocation, class C project LLM4SE (HP10C7ORC9). The collaboration is enabled by the NASA GSFC Internal Research and Development (IRAD) program, Deputy Director of the Engineering and Technology Directorate Dr. Joe Hill-Kittle, Astrophysics Division Chief Technologist Dr. Keith Jahoda, and the University of Maryland's Center for Research and Exploration in Space Science and Technology II (CRESST II) program.

References

[1] W. Alhindi, A. Aleid, I. Jenhani, and M. Mkaouer. 2023. IssueLabeler: an ALBERT-based Jira Plugin for Issue Classification.

In *2023 IEEE/ACM 10th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE Computer Society, Los Alamitos, CA, USA, 40–43. https://doi.org/10.1109/MOBILSoft59058.2023.00012

[2] Florian Angermeir, Maximilian Amougou, Mark Kreitz, Andreas Bauer, Matthias Linhuber, Davide Fucci, Fabiola Moyón C., Daniel Mendez, and Tony Gorschek. 2026. Reflections on the Reproducibility of Commercial LLM Performance in Empirical Software Engineering Studies. In *48th IEEE/ACM International Conference on Software Engineering, ICSE 2026*. Rio de Janeiro, Brazil, [TO APPEAR].

[3] Anthropic. 2024. Claude 3 Family. https://www.anthropic.com/news/claude-3-family Accessed: 2024-09-17.

[4] Giuliano Antoniol, Kamel Ayari, Massimiliano Di Penta, Foutse Khomh, and Yann-Gaël Guéhéneuc. 2008. Is it a bug or an enhancement? a text-based approach to classify change requests. In *Proceedings of the 2008 Conf. of the center for advanced studies on collaborative research: meeting of minds*. ACM, New York, NY, USA. https://doi.org/10.1145/1463788.1463819

[5] 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. https://doi.org/10.1145/3643787.3648043

[6] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. 2023. Qwen Technical Report. arXiv:2309.16609 [cs.CL] https://arxiv.org/abs/2309.16609

[7] Sebastian Baltes, Florian Angermeir, Chetan Arora, Marvin Muñoz Barón, Chunyang Chen, Lukas Böhme, Fabio Calefato, Neil Ernst, Davide Falessi, Brian Fitzgerald, Davide Fucci, Marcos Kalinowski, Stefano Lambiase, Daniel Russo, Mircea Lungu, Lutz Prechelt, Paul Ralph, Rijnard van Tonder, Christoph Treude, and Stefan Wagner. 2025. Guidelines for Empirical Studies in Software Engineering involving Large Language Models. *arXiv preprint arXiv:2508.15503* (2025). Available at `https://arxiv.org/abs/2508.15503`.

[8] Tegawendé F. Bissyandé, David Lo, Lingxiao Jiang, Laurent Réveillère, Jacques Klein, and Yves Le Traon. 2013. Got issues? Who cares about it? A large scale investigation of issue trackers from GitHub. In *2013 IEEE 24th Int'l Symposium on Software Reliability Engineering (ISSRE)*. `https://doi.org/10.1109/ISSRE.2013.6698918`

[9] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 1877–1901. `https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf`

[10] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang. 2023. Sparks of Artificial General Intelligence: Early experiments with GPT-4. arXiv:2303.12712 [cs.CL]

[11] Jialun Cao, Meiziniu Li, Ming Wen, and Shing chi Cheung. 2023. A study on Prompt Design, Advantages and Limitations of ChatGPT for Deep Learning Program Repair. arXiv:2304.08191 [cs.SE]

[12] Yiannis Charalambous, Norbert Tihanyi, Ridhi Jain, Youcheng Sun, Mohamed Amine Ferrag, and Lucas C. Cordeiro. 2023. A New Era in

Software Security: Towards Self-Healing Software via Large Language Models and Formal Verification. arXiv:2305.14752 [cs.SE]

[13] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374 [cs.LG]

[14] Jacob Cohen. 1968. Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit. *Psychological Bulletin* 70, 4 (1968), 213. https://doi.org/10.1037/h0026256

[15] Giuseppe Colavito, Filippo Lanubile, and Nicole Novielli. 2023. Few-Shot Learning for Issue Report Classification. In *2023 IEEE/ACM 2nd International Workshop on Natural Language-Based Software Engineering (NLBSE)*. 16–19. https://doi.org/10.1109/NLBSE59153.2023.00011

[16] Giuseppe Colavito, Filippo Lanubile, and Nicole Novielli. 2023. Issue Report Classification Using Pre-Trained Language Models. In *Proceedings of the 1st International Workshop on Natural Language-Based Software Engineering* (Pittsburgh, Pennsylvania) *(NLBSE '22)*. Association for Computing Machinery, New York, NY, USA, 29–32. https://doi.org/10.1145/3528588.3528659

[17] Giuseppe Colavito, Filippo Lanubile, and Nicole Novielli. 2025. Benchmarking large language models for automated labeling: The case of issue report classification. *Information and Software Technology* 184 (2025), 107758. https://doi.org/10.1016/j.infsof.2025.107758

[18] Giuseppe Colavito, Filippo Lanubile, Nicole Novielli, and Luigi Quaranta. 2024. Impact of data quality for automatic issue classification using pre-trained language models. *Journal of Systems and Software* 210 (2024), 111838. https://doi.org/10.1016/j.jss.2023.111838

[19] Giuseppe Colavito, Filippo Lanubile, Luigi Quaranta, and Nicole Novielli. 2024. Leveraging GPT-like LLMs to automate issue labeling. In *Proceedings of 21st International Conference on Mining Software Repositories (MSR 2024), April 2024* (Lisbon, Portugal). https://doi.org/10.1145/3643991.3644903

[20] Javier Luis Cánovas Izquierdo, Valerio Cosentino, Belén Rolandi, Alexandre Bergel, and Jordi Cabot. 2015. GiLA: GitHub label analyzer. In *2015 IEEE 22nd Int'l Conf.on Software Analysis, Evolution, and Reengineering (SANER)*. https://doi.org/10.1109/SANER.2015.7081860

[21] Tim Dettmers. 2021. bitsandbytes. https://github.com/TimDettmers/bitsandbytes Accessed: 2024-06-17.

[22] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient Finetuning of Quantized LLMs. In *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 10088–10115. https://proceedings.neurips.cc/paper_files/paper/2023/file/1feb87871436031bdc0f2beaa62a049b-Paper-Conference.pdf

[23] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. https://doi.org/10.18653/v1/N19-1423

[24] A. Fan, B. Gokkaya, M. Harman, M. Lyubarskiy, S. Sengupta, S. Yoo, and J. M. Zhang. 2023. Large Language Models for Software Engineering: Survey and Open Problems. In *2023 IEEE/ACM International*

Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE). IEEE Computer Society, Los Alamitos, CA, USA, 31–53. https://doi.org/10.1109/ICSE-FoSE59343.2023.00008

[25] Qiang Fan, Yue Yu, Gang Yin, Tao Wang, and Huaimin Wang. 2017. Where Is the Road for Issue Reports Classification Based on Text Mining?. In *2017 ACM/IEEE Int'l Symposium on Empirical Software Engineering and Measurement (ESEM)*. https://doi.org/10.1109/ESEM.2017.19

[26] GitHub. 2023. GitHub Copilot Now Has a Better AI Model and New Capabilities. https://github.blog/ai-and-ml/github-copilot/github-copilot-now-has-a-better-ai-model-and-new-capabilities/

[27] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. DeBERTa: Decoding-enhanced BERT with Disentangled Attention. arXiv:2006.03654 [cs.CL]

[28] Kim Herzig, Sascha Just, and Andreas Zeller. 2013. It's not a bug, it's a feature: How misclassification impacts bug prediction. In *2013 35th International Conference on Software Engineering (ICSE)*. 392–401. https://doi.org/10.1109/ICSE.2013.6606585

[29] Soneya Binta Hossain, Nan Jiang, Qiang Zhou, Xiaopeng Li, Wen-Hao Chiang, Yingjun Lyu, Hoan Nguyen, and Omer Tripp. 2024. A Deep Dive into Large Language Models for Automated Bug Localization and Repair. *Proc. ACM Softw. Eng.* 1, FSE, Article 66 (jul 2024), 23 pages. https://doi.org/10.1145/3660773

[30] Maliheh Izadi, Kiana Akbari, and Abbas Heydarnoori. 2022. Predicting the objective and priority of issue reports in software repositories. *Empirical Software Engineering* 2 (2022). https://doi.org/10.1007/s10664-021-10085-3

[31] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7B. arXiv:2310.06825 [cs.CL]

[32] Rafael Kallis, Andrea Di Sorbo, Gerardo Canfora, and Sebastiano Panichella. 2019. Ticket Tagger: Machine Learning Driven Issue Classification. In *2019 IEEE Intl. Conf. on Software Maintenance and Evolution, ICSME 2019, Cleveland, OH, USA, September 29 - October 4, 2019*. IEEE. https://doi.org/10.1109/ICSME.2019.00070

[33] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling Laws for Neural Language Models. arXiv:2001.08361 [cs.LG]

[34] Márk Lajkó, Viktor Csuvik, and László Vidács. 2022. Towards JavaScript program repair with Generative Pre-trained Transformer (GPT-2). In *2022 IEEE/ACM International Workshop on Automated Program Repair (APR)*. 61–68. https://doi.org/10.1145/3524459.3527350

[35] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. arXiv:1909.11942 [cs.CL]

[36] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Online, 7871–7880. https://doi.org/10.18653/v1/2020.acl-main.703

[37] Zhifang Liao, Dayu He, Zhijie Chen, Xiaoping Fan, Yan Zhang, and Shengzong Liu. 2018. Exploring the characteristics of issue-related behaviors in GitHub using visualization techniques. *IEEE Access* (2018). https://doi.org/10.1109/ACCESS.2018.2810295

[38] Marcos Macedo, Yuan Tian, Filipe Cogo, and Bram Adams. 2024. Exploring the Impact of the Output Format on the Evaluation of Large Language Models for Code Translation. In *Proceedings of the 2024 IEEE/ACM First International Conference on AI Foundation Models*

and Software Engineering (Lisbon, Portugal) *(FORGE '24)*. Association for Computing Machinery, New York, NY, USA, 57–68. https://doi.org/10.1145/3650105.3652301

[39] Meta. 2024. Meta LLaMA 3.1. https://ai.meta.com/blog/meta-llama-3-1/ Accessed: 2024-09-17.

[40] OpenAI. [n. d.]. API Documentation, How should I set the temperature parameter? https://platform.openai.com/docs/guides/text-generation/how-should-i-set-the-temperature-parameter. Accessed: 2023-11-17.

[41] OpenAI. 2022. ChatGPT: Optimizing Language Models for Dialogue. https://platform.openai.com/docs/models#gpt-3-5-turbo

[42] OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]

[43] Sebastiano Panichella, Gabriele Bavota, Massimiliano Di Penta, Gerardo Canfora, and Giuliano Antoniol. 2014. How Developers' Collaborations Identified from Different Sources Tell Us about Code Changes. In *2014 IEEE International Conference on Software Maintenance and Evolution*. https://doi.org/10.1109/ICSME.2014.47

[44] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training. (2018). https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

[45] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *J. Mach. Learn. Res.* 21, 1, Article 140 (jan 2020), 67 pages. https://jmlr.org/papers/volume21/20-074/20-074.pdf

[46] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (Eds.). Association for Computational Linguistics, Hong Kong, China, 3982–3992. https://doi.org/10.18653/v1/D19-1410

[47] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2020. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv:1910.01108 [cs.CL]

[48] Fabrizio Sebastiani. 2002. Machine learning in automated text categorization. *ACM Comput. Surv.* 34, 1 (2002), 1–47. https://doi.org/10.1145/505282.505283

[49] D. Sobania, M. Briesch, C. Hanna, and J. Petke. 2023. An Analysis of the Automatic Bug Fixing Performance of ChatGPT. In *2023 IEEE/ACM International Workshop on Automated Program Repair (APR)*. IEEE Computer Society, Los Alamitos, CA, USA, 23–30. https://doi.org/10.1109/APR59189.2023.00012

[50] Gemini Team. 2024. Gemini: A Family of Highly Capable Multimodal Models. arXiv:2312.11805 [cs.CL] https://arxiv.org/abs/2312.11805

[51] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288 [cs.CL]

[52] Lewis Tunstall, Edward Beeching, Nathan Lambert, Nazneen Rajani, Kashif Rasul, Younes Belkada, Shengyi Huang, Leandro von Werra, Clémentine Fourrier, Nathan Habib, Nathan Sarrazin, Omar Sanseviero,

Alexander M. Rush, and Thomas Wolf. 2023. Zephyr: Direct Distillation of LM Alignment. arXiv:2310.16944 [cs.LG]

[53] Lewis Tunstall, Nils Reimers, Unso Eun Seo Jo, Luke Bates, Daniel Korat, Moshe Wasserblat, and Oren Pereg. 2022. Efficient Few-Shot Learning Without Prompts. https://doi.org/10.48550/arXiv.2209.11055

[54] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[55] Anthony J Viera and Joanne M Garrett. 2005. Understanding interobserver agreement: the kappa statistic. *Family medicine* (2005). https://pubmed.ncbi.nlm.nih.gov/15883903/

[56] Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang. 2024. Software Testing With Large Language Models: Survey, Landscape, and Vision. *IEEE Trans. Softw. Eng.* 50, 4 (Feb. 2024), 911–936. https://doi.org/10.1109/TSE.2024.3368208

[57] Jun Wang, Xiaofang Zhang, and Lin Chen. 2021. How well do pre-trained contextual language representations recommend labels for GitHub issues? *Knowledge-Based Systems* (2021). https://doi.org/10.1016/j.knosys.2021.107476

[58] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. 2022. Emergent Abilities of Large Language Models. arXiv:2206.07682 [cs.CL]

[59] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing*

Systems, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 24824–24837. https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf

[60] Frank F. Xu, Uri Alon, Graham Neubig, and Vincent Josua Hellendoorn. 2022. A Systematic Evaluation of Large Language Models of Code. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming* (San Diego, CA, USA) *(MAPS 2022)*. Association for Computing Machinery, New York, NY, USA, 1–10. https://doi.org/10.1145/3520312.3534862

[61] Zhen Yang, Fang Liu, Zhongxing Yu, Jacky Wai Keung, Jia Li, Shuo Liu, Yifan Hong, Xiaoxue Ma, Zhi Jin, and Ge Li. 2024. Exploring and Unleashing the Power of Large Language Models in Automated Code Translation. *Proc. ACM Softw. Eng.* 1, FSE, Article 71 (jul 2024), 24 pages. https://doi.org/10.1145/3660778

[62] Burak Yetistiren, Isik Ozsoy, and Eray Tuzun. 2022. Assessing the quality of GitHub copilot's code generation. In *Proceedings of the 18th International Conference on Predictive Models and Data Analytics in Software Engineering* (Singapore, Singapore) *(PROMISE 2022)*. Association for Computing Machinery, New York, NY, USA, 62–71. https://doi.org/10.1145/3558489.3559072

[63] Hao Yu, Bo Shen, Dezhi Ran, Jiaxin Zhang, Qi Zhang, Yuchi Ma, Guangtai Liang, Ying Li, Qianxiang Wang, and Tao Xie. 2024. CoderEval: A Benchmark of Pragmatic Code Generation with Generative Pre-trained Models. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (Lisbon, Portugal) *(ICSE '24)*. Association for Computing Machinery, New York, NY, USA, Article 37, 12 pages. https://doi.org/10.1145/3597503.3623316

[64] Zhiqiang Yuan, Junwei Liu, Qiancheng Zi, Mingwei Liu, Xin Peng, and Yiling Lou. 2023. Evaluating Instruction-Tuned Large Language Models on Code Comprehension and Generation. arXiv:2308.01240 [cs.CL]

[65] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang,

Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2023. A Survey of Large Language Models. arXiv:2303.18223 [cs.CL]

[66] Yu Zhou, Yanxiang Tong, Ruihang Gu, and Harald Gall. 2016. Combining text mining and data mining for bug report classification. *J. Softw. Evol. Process* 28, 3 (March 2016), 150–176. `https://doi.org/10.1002/smr.1770`

[67] Liu Zhuang, Lin Wayne, Shi Ya, and Zhao Jun. 2021. A Robustly Optimized BERT Pre-training Approach with Post-training. In *Proceedings of the 20th Chinese National Conference on Computational Linguistics*, Sheng Li, Maosong Sun, Yang Liu, Hua Wu, Kang Liu, Wanxiang Che, Shizhu He, and Gaoqi Rao (Eds.). Chinese Information Processing Society of China, Huhhot, China, 1218–1227. `https://aclanthology.org/2021.ccl-1.108`