Leveraging Large Language Models for Sentiment Analysis in GitHub Pull Request Discussions

Anonymous Authors

Abstract*—Collaborative platforms like GitHub and GitLab are essential in modern software development, enabling asynchronous, distributed work through features such as pull requests. These pull requests foster discussions that address various aspects of code, requirements, and design, which are crucial for long-term software quality. However, traditional sentiment analysis tools underperform in capturing the nuances of sentiment within software engineering texts, such as the technical jargon, context-specific expressions, and multi-faceted discussions present in pull request comments, making it difficult to gauge the true emotional tone of developer communication, limiting their utility for insights into developer communications. This has led to the creation of software engineering specific tools, but recent studies have also observed that they underperform. Thus, this study explores the potential of large language models, which have shown a greater ability to understand context and domain-specific language nuances, to enhance sentiment analysis in this context by evaluating eight different LLMs, including proprietary and open-source models, across 32 configurations using four prompt engineering techniques. Using a dataset of 1,791 pull request messages from 36 GitHub projects, we analyze trade-offs in performance, cost, and deployment feasibility for each model. Our findings indicate that while no single model excels universally, each demonstrates strengths in specific areas, and other aspects such as task requirements, cost, and the availability of resources should be prioritized when choosing a model. Additionally, we also observed that larger models benefiting significantly from prompt-engineering techniques like chain-of-thought. These results underscore the value of tailored LLM selection and prompt engineering to build effective sentiment analysis tools for collaborative software platforms. This work provides actionable insights for researchers, tool builders, and practitioners looking to leverage LLMs for sentiment analysis in developer communication, showcasing the potential to enhance collaboration and software quality through more accurate sentiment insights.

***Index Terms*—repository mining, large language models, human aspects, sentiment analysis**

I. Introduction

Modern software development relies on cloud-based collaborative platforms such as GitHub and GitLab[1], which provide social media features that are essential to enable the asynchronous and distributed development process that is extremely common nowadays [1], [2]. One of these features, pull requests, allows developers to propose changes to a system and discuss them via a messaging system [3]. These messages can be related to a specific excerpt of code, or can also be of a more broad nature [4]. Recent studies have related this latter type of message with discussions about requirements [5] and design decisions [6], which can dictate future quality aspects of a project [7].

Previous research has indicated that human aspects related to communication in cloud-based collaborative platforms can significantly impact the software development process, from influencing the introduction of bugs to even the acceptance of a pull request itself [8], [9]. In this context, understanding how this communication occurs and how it can be most effective becomes a crucial target for practitioners and researchers [10]. Although there is a myriad of methods that can be used to increase this understanding, one venue that has been frequently explored, and associated with different aspects of the development process, is sentiment analysis[2] expressed in this communication [12], [13].

Since general-purpose sentiment analysis tools are known to perform poorly in software engineering-related texts [14], several tools specific to software engineering have been built [15], [16]. However, a recent evaluation of these tools highlighted that, besides often being difficult to execute due to several reasons (*e.g.*, broken links, insufficient documentation, requiring outdated tools, etc.), their performance[3] is unsatisfactory in the specific context of pull request discussions [17], [18]. This scenario makes the development of new tools utilizing large language models (LLMs) appealing, given their ability to understand and incorporate domain-specific nuances (*i.e.*, technical language and terminology) [19], [20].

The goal of this work is, therefore, to evaluate the performance and practicality of eight LLMs for sentiment analysis in the context of GitHub, the biggest collaborative development platform in the world, focusing primarily on pull request discussions. Specifically, we aim to identify how well LLMs can classify sentiment in software engineering texts, and examine the impact of different prompt engineering techniques on their performance metrics. By comparing eight LLMs, including both closed and open-source options, we assess trade-offs in accuracy, cost, and deployment feasibility. This analysis should provide insight into whether certain LLMs, particularly when combined with tailored prompt engineering, can effectively handle the unique language and sentiment patterns of developer communications. By investigating this, we believe our findings offer guidance on selecting models based on

[1]https://github.com/ and https://gitlab.com/.

[2]In this paper, we define *Sentiment* as the focus of *Sentiment Analysis*, which, in this context, involves identifying a classification that summarizes the subjective elements of a sentence—such as opinions or emotions. It may be positive, negative, or neutral [11].

[3]In this work, we refer to performance as the precision, recall and f1-score metrics that a specific tool reaches.

specific task needs, cost constraints, and implementation goals, advancing the development of accurate and scalable sentiment analysis tools within collaborative software environments. To perform this our study, we rely on a dataset [21] containing 1,791 labeled pull request discussion messages from 36 GitHub projects.

In our results, we observed that all models outperformed traditional sentiment analysis tools (considering the results of an evaluation from the literature [21]), with notable variability in strengths across different sentiment classes (positive, negative, neutral). The closed-source model GPT-4o achieved the best overall balance, particularly benefiting from advanced prompt engineering techniques, such as chain-of-thought, which improved precision and recall metrics. Furthermore, open-source models, including Mistral and Llama3.1, demonstrated competitive performance with distinct advantages in scalability, cost, and flexibility. Furthermore, we also discuss how the difference between closed vs. open-source models can affect additional aspects such as privacy, when deployed locally. While larger models generally benefited more from prompt-engineering techniques, we found that no single LLM excelled universally, underscoring the importance of selecting models based on specific task requirements, cost considerations, and implementation constraints (e.g., hardware availability). These findings indicate that LLMs, particularly with prompt engineering, present a promising path forward for building sentiment analysis tools in software development. As such, the contributions of this work are as follows:

- A comprehensive study of how eight different LLMs perform when evaluating the sentiment of pull request messages on GitHub.
- A detailed discussion about the practicality of a variety of ways of selecting and deploying (both in terms of cost and performance) LLMs (*e.g.,* closed-source vs. open-source, cloud vs. local).
- An in-depth analysis on how prompt engineering techniques can be applied in sentiment analysis applications and how they affect performance metrics.
- We also provide actionable implications for researchers, tool builders, and practitioners on how they can select and utilize a model for sentiment analysis (see Section VI). For example, we describe how no model was universally better, and that instead of looking only at performance, other aspects such as task requirements and cost should probably be prioritized. We also discuss how prompt engineering has diminishing returns, especially when working with smaller models.

Audience: Researchers, practitioners, and tool builders benefit from our experiments and insights from different perspectives (e.g., how some of our discussion on how executing models locally might affect privacy might only be relevant for specific contexts). Our analyses, and subsequent findings, can be utilized in understanding how effective LLMs are for sentiment analysis on discussions mined from software projects, such as GitHub pull request discussions. Furthermore, our work showcases the potential of LLMs at sentiment analysis on messages from other sources such as Slack, Gitter, forums, among others.

II. Background and Related Work

A. Sentiment Analysis

Sentiment analysis, often referred to as opinion mining, involves detecting emotions, feelings, and opinions [22]. As a branch of natural language processing (NLP), it aims to identify and measure the polarity (positive, negative, neutral) and emotional tone of textual data [23]. Traditionally, sentiment analysis relied on either rule-based or learning-based approaches. Rule-based approaches apply predefined linguistic rules and lexicons to interpret sentiment, whereas learning-based approaches (*e.g.*, Naive Bayes and Support Vector Machines) learn from labeled data to classify sentiments [24]. Although these traditional methods can achieve some success, they often struggle with complex contexts, including sarcasm and idiomatic expressions [24].

In software engineering, understanding sentiment across team members is essential, as sentiments directly correlate with productivity and product quality [25], [26]. Sentiment analysis can serve as a powerful feedback tool in software project management by assessing textual interactions, such as chat messages, code review comments, and other communications within version control systems. By detecting patterns of negative sentiment, project managers can proactively address emerging issues, helping to prevent conflicts and reduce team stress. Promoting a constructive atmosphere not only boosts team morale, but also enhances innovation and creativity, essential drivers in software development [27]. By systematically analyzing and managing team members' sentiment, software engineering teams can strengthen communication, enhance collaboration, and improve the overall well-being of team members, all critical elements for achieving project success [27], [28].

B. LLMs in Sentiment Analysis

The rise of LLMs, such as transformer-based models like BERT [29] and GPT [19], has revolutionized natural language processing (NLP). These LLMs are built on the transformer architecture [30], which uses self-attention mechanisms to capture relationships between words in a text, regardless of their position. This enables them to grasp context and nuance far more effectively than earlier approaches, such as bag-of-words or basic word embeddings. One of the core strengths of LLMs is their ability to generalize across domains and tasks with minimal fine-tuning, making them exceptionally powerful for tasks such as sentiment analysis, where subtle variations in language and context are critical [31]. Advanced LLMs, like GPT-4 [19] are pre-trained on massive datasets, enabling them to generate human-like responses and perform a wide range of NLP tasks. By applying techniques such as prompt engineering, these models can be further refined to capture fine-grained nuances and complex contexts, enhancing both the accuracy and robustness of their analyses [31]–[34].

To use the maximum capacity of LLMs, accurate prompts are essential. Prompt engineering refers to the process of designing specific input prompts to elicit the desired responses from LLMs. The effectiveness of LLMs in a particular task can be heavily influenced by how prompts are structured [35]. For instance, incorporating additional contextual details in prompts allows models to refine their responses, leading to more precise sentiment classification and extraction of nuanced emotional cues from text. Techniques such as few-shot learning, where a few task examples are included within the prompt, allow the model to adapt to new tasks without extensive fine-tuning [19]. This is particularly important in sentiment analysis, where subtle differences in prompt design can lead to varying interpretations of emotional tone or sentiment. Properly crafted prompts can ensure that LLM focuses on relevant aspects of the text, leading to more accurate and reliable sentiment analysis [34]. This approach enhances the model's ability to manage the inherent complexity and ambiguity in human language, which is especially valuable in domains like software engineering communication, where context is paramount [21].

C. Related Work

Traditional sentiment analysis techniques, such as lexicon-based approaches and classical machine learning models (*e.g.*, [36]–[38]), have several limitations. Lexicon-based approaches often struggle to capture the contextual meaning of words, especially when dealing with domain-specific language or idiomatic expressions [24]. Similarly, classical machine learning models are typically less effective at handling ambiguous or context-dependent language without extensive feature engineering [23].

These limitations become especially pronounced in the field of software engineering, where communication often includes technical jargon, informal language, and sarcasm. LLMs, such as GPT-4, with their ability to model long-range dependencies and capture nuanced meaning, offer a significant improvement over traditional methods in performing sentiment analysis [31]. GPT-4 outperforms smaller models like BERT or seBERT in scenarios with limited labeled data due to its flexibility in handling diverse linguistic inputs with minimal training [39], [40]. This practical advantage makes GPT-4 highly suitable for software engineering, where manually labeling data is time-consuming and labor-intensive.

While GPT-4 offers significant advantages, its status as a paid solution introduces several drawbacks. For organizations needing to deploy sentiment analysis at scale, using GPT-4 can become very expensive. To overcome such limitations, open LLMs can be used. They can be combined with prompt engineering strategies that can further amplify their ability to tackle complex domains like software engineering without extensive retraining. Recent studies [33], [34], [39] demonstrate that well-crafted prompts can significantly boost LLM performance in sentiment analysis by enhancing model interpretability and relevance. Xing et al. [33] proposed a framework where heterogeneous LLM agents collaborate, discussing specific sentiment classification tasks to arrive at a more robust analysis with minimal data. Additionally, Wei et al. [34] emphasize the value of prompt engineering for structured information extraction tasks. These strategies become especially valuable when handling complex, unstructured data sources, such as discussions in collaborative development platforms, where understanding context and disambiguating language are essential for accurate sentiment interpretation, especially in cross-platform scenarios (*e.g.*, GitHub and Jira) [41].

Many studies in the field of sentiment analysis within software engineering have primarily focused on aspects such as developer communications in issue trackers, forums, or chat rooms [16], [42]. On one of these platforms, GitHub, one critical element that is often looked at is pull requests (PRs). Previous works often look at PRs only as a venue for studying code review [43], overlooking that there a PR also contains a general discussion thread, that is often not directly related to a specific excerpt of source code. Recent studies suggest that this thread (which we call pull request discussion) is where higher level design decisions about the source code are often made [7], [44], as they are a place for discussing higher-level topics, relating to the whole change being proposed as part of the pull request.

Studies in the literature (e.g., [16], [45]) indicate that emotions expressed in software development discussions can affect the quality of decisions and overall productivity. Thus, incorporating sentiment analysis on pull request discussions allows for a richer understanding of the social dynamics at play and can inform interventions to improve collaboration within open-source software projects. While there are some tools that are specifically tailored for this type of discussion scenario [13], a recent study [21] has revealed that available tools, while usable in this context, do not perform well. Therefore, our hypothesis is that utilizing LLMs we can potentially have a more nuanced understanding of the technical language that is employed in these discussions, for this task, can have promising results.

III. Study Design

This study aims *to investigate the effectiveness, cost, and applicability of LLMs for executing sentiment analysis tasks, focusing specifically on the context of pull request discussions on GitHub*. Given the specialized language and sentiment nuances in developer communications, our goal is to determine whether LLMs can reach higher accuracy than traditional sentiment analysis tools and to understand how prompt engineering might improve their performance. We chose to compare several models (and different variations, in terms of model complexity) that cover a wide range of requirements (*e.g.*, open-source vs. proprietary, local vs. cloud-hosted, higher vs. lower hardware requirements, etc.).

A. Research Questions

Base on our aim with our work, this study is guided by three research questions (RQs), as follows:

$\mathbf{RQ_1}$: What is the performance of our baseline LLM, GPT-4o, while executing sentiment analysis tasks?

We start by establishing a performance baseline using GPT-4o, a widely used proprietary LLM, to set a reference point for evaluating other models. This baseline enables us to assess how well LLMs handle the unique challenges of sentiment analysis in developer discussions and allows for comparisons in performance metrics and applicability across models.

$\mathbf{RQ_2}$: To what extent do other LLMs differ from our baseline for executing the same tasks, in terms of cost and performance?

Building on our baseline, we start comparing other LLMs available currently, both proprietary and open-source, to examine variations in cost, scalability, and performance. This comparative analysis includes a focus on trade-offs relevant to practical deployment, such as the need for hardware resources or reliance on cloud infrastructure. While the models will be evaluated in terms of performance metrics (*i.e.*, precision, recall, and F1-score), we also provide actionable insights into the advantages and limitations of each model type based on deployment needs and resource constraints.

$\mathbf{RQ_3}$: How does the usage of prompt engineering techniques affect the performance of large language models in sentiment analysis tasks?

A common strategy to optimize the performance of LLMs is by using prompt engineering techniques, which are known to achieve better results in certain tasks. Therefore, we experimented several prompt engineering techniques that are suitable for the task of software engineering, to make sure that we are effectively and fairly evaluating the models. We also investigate whether these techniques could possibly be a way to bridge the gap between cheaper and more performant (in terms of the precision, recall, and F1-score) models.

B. Study Steps

Our study is composed of five steps, which are illustrated in Figure 1. They are also described as follows.

1) Dataset Selection. For this study, we selected the PRemo dataset [21], which comprises 1,791 pull request messages manually labeled for sentiment by 19 experts. This dataset was chosen because it aligns with our focus on GitHub pull request discussions. Also, it is grounded in an emotion model from psychology, with the labeling process being conducted by both neuroscientists and software engineers. Although the dataset includes preprocessed versions of the messages, we opted to use the raw, unprocessed data. A pilot test with preprocessed messages showed no significant differences in the results.

2) LLM Selection. For selecting the LLMs for this work, we considered criteria such as availability, popularity, and scalability. First, in terms of availability, we had access to one cloud provider during the execution of this study, namely OpenAI. As such, we selected the flagship GPT4o available for this provider, which was chosen as the baseline for RQ_1. For other models, since one of our objectives is to discuss the advantages of different ways of deploying these models, we opted to utilize the Ollama[4] tool to deploy Llama 3.1 (from Meta), and Mistral and Gemma (from Google). In terms of popularity, all the chosen families of models (except for GPT4o, which is closed source, but widely utilized in previous studies [46]) are part of the top 10 most popular (in terms of downloads) families of LLMs available on hugging face[5]. Finally, regarding scalability, all models chosen are available in different variations, which mainly affect parameter count. The parameter count of a model dramatically affects its performance, especially inference speed and memory requirements. A higher parameter count can also be associated with higher reasoning capabilities. The names and characteristics of all models chosen are available in Table I.

TABLE I
OVERVIEW OF THE LARGE LANGUAGE MODELS EVALUATED IN THIS WORK

Model Name	Parameters	Release Date	Provider	Quantization
GPT 4o	-	May 13, 2024	OpenAI	-
GPT 4o mini		July 18, 2024		
Mistral Nemo	12 billion	June 18, 2024	Ollama	Q4_0
Mistral Small	22 billion	September 17, 2024		
Gemma 2	9 billion	June 27, 2024	Ollama	Q4_0
	27 billion			
Llama3.1	8 billion	July 23, 2024	Ollama	Q4_0
	70 billion			

3) Prompt Engineering. In terms of prompt engineering, we started with a base prompt, that was utilized as a baseline for RQ_1.[6] For RQ_3, we selected a number of prompt engineering techniques that have been utilized in previous works to successfully improve the results of classification tasks [48]. As such, the prompt engineering techniques chosen are related to how we can provide examples to the LLM. We selected three techniques, which are: (i) one-shot, providing one example to the LLM; (ii) few-shot, providing three or more examples to the LLM; and (iii) chain of thought, providing one or more examples to the LLM, which contain the reasoning behind each response. We experimented with more than one way of executing each of these techniques, which is discussed in Section IV.

4) Model Execution. As aforementioned, we mainly utilized two providers: OpenAI (cloud-based) and Ollama (local) to execute the LLMs chosen. The locally-deployed LLMs were executed in a computer with the following specifications: i9-13900HX processor, 32GB of RAM and an NVIDIA RTX 4090 Laptop GPU, with 16GB of VRAM. In terms of model configuration, since sentiment analysis is as a classification

[4]https://ollama.com/

[5]https://huggingface.co/models?pipeline_tag=text-generation&sort=downloads

[6]All prompts utilized in this work are available in the scripts contained in the replication package [47].

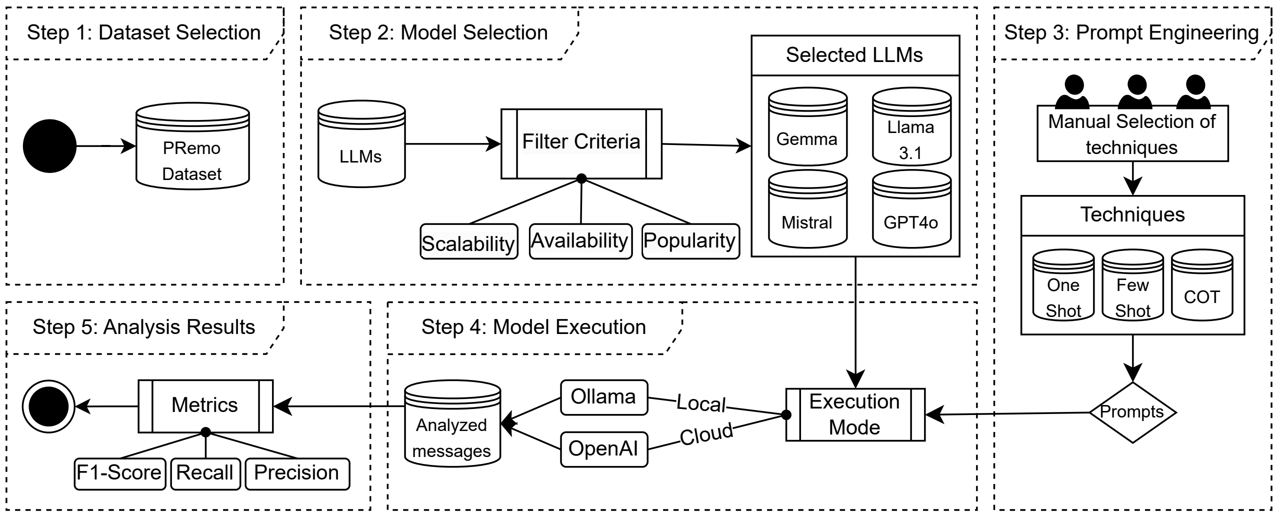

Fig. 1. Overview of the Study Steps

TABLE II
PERFORMANCE FOR OUR BASELINE MODEL: GPT 4O (N=1791)

	Precision	Recall	f1-score
Positive	93%	46%	0.61
Negative	78%	40%	0.53
Neutral	59%	93%	0.73
Macro Avg.	77%	60%	0.62
Micro Avg.	67%	0.67%	0.67

task, we can aim for reproducibility by tuning the model to be as deterministic as possible, thus removing the need for multiple runs using each prompt.[7] This can be achieved by tuning the *temperature* parameter, which was set to 0 throughout the entire execution of this study.

5) Analysis of the Results. We utilized Python scripts (available as part of the replication package [47]) that leveraged the pandas library[8] to analyze the output of the LLMs in terms of precision, recall, and F1-score. Since depending on the goal of the user when utilizing sentiment analysis, either metrics can be relevant. We discuss the results in terms of different possible usage scenarios (see Section IV-A).

IV. RESULTS AND DISCUSSION

The following subsections present the results for each RQ. Based on the results, we highlight the most important findings.

A. Performance of our baseline model, GPT-4o (RQ_1)

Table II summarizes the performance metrics (Precision, Recall, and F1-score) for our baseline model, GPT-4o. In summary, for positive sentiment, the model achieves high precision (0.93) but low recall (0.46), indicating that GPT-4o accurately identifies positive messages but often misses them. In negative sentiment, the precision is moderate (0.78), which means it is adequate at capturing negative sentiment without mislabeling it. Yet, the low recall (0.40) suggests that GPT-4o misses substantial negative messages. Neutral sentiment shows high recall (0.93) with a lower precision (0.59), meaning the LLM correctly identifies most neutral messages but frequently misclassifies others as neutral as well. The micro and macro averages across metrics illustrate a balanced overall performance but underscore the need for improved recall in critical classes.

Evaluating whether the performance of our baseline tool is good depends on various factors, the most important of which is the task for which a model will be employed. Some examples of how tasks can affect the prioritization of performance metrics are in what follows.

Toxicity Detection in Issue Comments or Pull Requests: For toxicity detection, it is crucial to minimize false positives (*i.e.*, incorrectly tagging non-toxic comments as toxic). High precision in the negative class helps reduce such false positives, ensuring that comments are only flagged as toxic when there is high confidence. This is essential to prevent mislabeling constructive criticism or neutral comments as negative, which could discourage constructive feedback and impact developer collaboration. Utilizing a bot to moderate comments in issues and pull requests could also be a similar example.

Identifying Informative Feedback for Prioritization: When looking to gather all constructive or positive feedback for project prioritization or improvement, recall in the positive class becomes essential. A higher recall ensures that the model captures as many relevant comments as possible, reducing the risk of overlooking valuable feedback. In this case, false

[7]While some randomness in intrinsic to LLMs (given different hardware and software implementation details) we experimented with different parameters and judged that the randomness between our runs using *temperature=0* was negligible.

[8]https://pandas.pydata.org/

negatives (missing constructive comments) are more costly than false positives, as missing input could result in critical improvement opportunities being ignored.

Detecting Neutral or Information-Only Comments: In a task where the goal is to identify comments that are informational or non-opinionated (neutral), balancing precision and recall is crucial. The F1-score for the neutral class allows the model to consistently capture neutral comments while minimizing both false positives and false negatives. This is especially useful in contexts where developers want to differentiate between sentiment-laden feedback (positive or negative) and purely informational updates.

Automated Sentiment-Based Assignment for Support Teams: In cases where negative sentiment indicates a need for escalation to support or development teams, recall in the negative class is important. High recall ensures that nearly all dissatisfied or problematic comments are captured, allowing the support team to address issues before they escalate. Missing out on negative comments (*i.e.*, false negatives) could mean leaving critical issues unaddressed, potentially damaging user satisfaction and trust.

Sentiment-Based Trend Analysis Over Time: For more classical research purposes, that tracks the overall sentiment in project comments over time, a balanced performance across all classes is ideal.

These different priorities mean that a number of metrics need to be considered when discussing the applicability of a model. Nevertheless, while improving the recall would be desirable, the baseline results already outperform previous techniques utilized in the literature. An evaluation using the same dataset found that the best performing tool proposed in the literature only had 0.57 precision when identifying negative messages (see Table III). In any case, a more nuanced discussion of the performance of this specific model can only be done by comparing it to other LLMs, which is presented in the next subsections.

Finding 1: GPT-4o, our baseline model, shows high precision in positive and negative sentiment, but lacks recall, particularly for negative sentiment. Nevertheless, for applications in the context of pull request discussions, it already outperforms most previous techniques.

B. *Performance of other models (RQ_2)*

Table IV summarizes the performance metrics (Precision, Recall, and F1-score) for the additional seven LLMs investigated in this work.

The first model we experimented for this RQ was GPT-4o mini. Before discussing its performance metrics, we discuss its applicability for building sentiment analysis tools. Closed-source models have all sorts of advantages when building tools, most of which come from the fact that often (and this is the case for the GPT family of models) they are executed utilizing cloud-based APIs. Thus, most of the processing is offloaded to the cloud. This also means that most of the overhead of running, maintaining, and deploying the model is responsibility of the provider, allowing developers and users of the tool to focus only on using it. This model of development, however, also comes with some downsides. When utilizing these APIs, developers lose control of the privacy of your data, as they become dependent on the cloud provider for uptime and availability (*e.g.*, the specific version of the model one utilizes might be discontinued). Finally, while the upfront hardware costs are not necessary, utilizing the cloud means that companies need to constantly pay for utilization, and costs can quickly accumulate, especially when utilizing expensive models, such as our baseline GPT-4o.

TABLE III
PERFORMANCE METRICS FOR NON-LLM TOOLS AVAILABLE IN THE LITERATURE. REPORTED BY [21] UTILIZING THE SAME DATASET AS THIS STUDY AS A BENCHMARK.

Class	Tool	Precision	Recall	f1-score
Positive	SentiStrength	55.32%	**78.89%**	0.65
	SentiStrengthSE	**79.23%**	62.96%	**0.7**
	DEVA	66.49%	72.36%	0.69
	Senti4SD	56.47%	71.21%	0.63
Negative	SentiStrength	45.06%	69.68%	0.55
	SentiStrengthSE	43.56%	**79.86%**	0.56
	DEVA	48.41%	70.37%	**0.57**
	Senti4SD	**57.32%**	42.59%	0.49
	SentiCR	42.79%	39.81%	0.41
Neutral	SentiStrength	**80.53%**	36.52%	0.5
	SentiStrengthSE	79.32%	55.37%	0.65
	DEVA	79.87%	56.80%	**0.66**
	Senti4SD	65.81%	**63.84%**	0.65
Macro Avg.	SentiStrength	60.30%	61.70%	0.61
	SentiStrengthSE	**67.37%**	66.06%	**0.67**
	DEVA	59.87%	59.21%	0.6
	Senti4SD	64.92%	**66.51%**	0.66

In the case of OpenAI, one of the ways to mitigate some of the drawbacks when utilizing their models is by providing a lighter, cheaper model, which is GPT-4o mini. In our evaluation, GPT-4o mini performed comparably to the baseline model (*i.e.*, GPT-4o). However, it performed much worse in terms of recall for the negative class, with a difference of 12%. Nonetheless, we can notice that the baseline was already not satisfactory, with only 40% recall. As mentioned in Section IV-A, the negative class is critical when utilizing sentiment analysis in software engineering. On the other hand, it had much better recall for the positive class than the baseline model, namely a difference of 20%. In sum, the results for closed-source models are as follows.

Finding 2: GPT-4o mini is overall less balanced than GPT-4o, but the performance is comparable, and should be preferred in situations where capturing positive sentiment is important (see Section IV-A for examples).

The remainder of the LLMs experimented in our study

TABLE IV
PERFORMANCE METRICS FOR ALL MODELS TESTED, EXCEPT FOR THE BASELINE MODEL

Model	Class	Precision	Recall	f1-score
GPT-4o mini (n = 1791)	Positive	0.82	0.66	0.73
	Negative	**0.77**	0.28	0.41
	Neutral	0.61	0.89	**0.73**
	Micro Avg.	0.74	0.61	0.62
	Macro Avg.	0.68	0.68	0.68
Mistral Small (n = 1790)	Positive	**0.92**	0.51	0.66
	Negative	0.72	0.42	0.53
	Neutral	0.60	**0.90**	0.72
	Macro Avg.	**0.75**	0.61	0.63
	Micro Avg.	0.67	0.67	0.67
Mistral Nemo (n = 1783)	Positive	0.77	0.74	0.75
	Negative	0.67	0.51	0.58
	Neutral	0.67	0.76	0.71
	Macro Avg.	0.7	**0.67**	**0.68**
	Micro Avg.	**0.69**	**0.69**	**0.69**
Gemma2 27B (n = 1790)	Positive	0.64	0.83	0.72
	Negative	0.68	0.52	**0.59**
	Neutral	0.69	0.65	0.67
	Macro Avg.	0.67	**0.67**	0.66
	Micro Avg.	0.67	0.67	0.67
Gemma2 9B (n = 1790)	Positive	0.77	0.69	0.73
	Negative	0.68	0.44	0.53
	Neutral	0.64	0.79	0.70
	Macro Avg.	0.69	0.64	0.66
	Micro Avg.	0.68	0.68	0.68
Llama 3.1 8B (n = 1791)	Positive	0.55	**0.85**	0.67
	Negative	0.60	**0.56**	0.58
	Neutral	**0.71**	0.49	0.58
	Macro Avg.	0.62	0.64	0.61
	Micro Avg.	0.61	0.61	0.61
Llama3.1 70B (n = 1791)	Positive	0.74	0.78	**0.76**
	Negative	0.70	0.40	0.51
	Neutral	0.66	0.78	0.71
	Macro Avg.	**0.75**	0.61	0.63
	Micro Avg.	0.67	0.67	0.67

are open-source models that were executed locally. Before discussing how these, often smaller, open-source models can compare with the closed-source LLMs, we return to the discussion for proprietary vs. open-source models. Open-source LLMs forgo some positive points discussed for proprietary models, while gaining some advantages in return. For example, there is an upfront hardware requirement needed to run these models locally (*e.g.*, the version of Llama3.1 70B utilized in this work utilized approx. 40GB of system memory). However, after the hardware is paid for, there are no additional fees for using the model, improving therefore the scalability of using them. This approach of running the model locally also has advantage in terms of privacy, since the data does not need to be sent through an external API controlled by another company. These advantages all come from the additional control you have for deploying and tuning these LLMs. Finally, this locally-deploy approach is also optional, and there are several cloud providers that offer APIs for running these models with the same aforementioned advantages that closed-source models have. This means that a model can be firstly tested locally, and then deployed in the cloud. This flexibility is ideal for building open-source tools that leverage LLMs. We take this aspect into consideration when evaluating which model is more effective (both in terms of performance and cost) when building a sentiment analysis tool.

For the performance metrics of the LLMs, Llama3.1 70B achieves the highest F1-score for the positive class (0.76) while maintaining a good balance across all metrics, making it suitable for tasks requiring reliable general performance. In contrast, GPT-4o mini and Mistral Small achieved the highest precision in detecting negative content (0.77 and 0.72, respectively) but did not reach better precision than the baseline model. They also suffer from low recall for the negative class. However, since applications that deal with the negative class tend to favor precision (*i.e.*, it is preferable to not flag a message as negative, than to flag it incorrectly), as we discuss in the examples in Section IV-A, this problem is not critical. The most balanced of the models were Mistral Nemo and Gemma2 27B. The latter had an F1 score of 0.67 in the neutral class, showing suitability for applications such as capturing informational messages.

Finding 3: While the most balanced of the models was our baseline model, namely GPT-4o, the other models that are much smaller and cheaper, managed to compete (and ever overcome it) in several tasks. Thus, these other LLMs proved their applicability in building tools for analysis software engineering messages in the context of pull request discussions.

As aforementioned, different LLMs may serve different user requirements depending on the task at hand. For instance, if the task is toxicity detection, the model's precision and recall in the negative class become crucial. Mistral Small and Gemma2 27B show balanced negative class precision and recall, while Llama3.1 8B maximizes recall (0.56), though it compromises on precision. Thus, the best model choice depends on the prioritization of specific metrics (*e.g.*, precision for false positive control, or recall for sensitivity), task requirements, and associated cost constraints for optimal resource allocation. Overall, while no single model excels universally, each alternative demonstrates distinct strengths in certain areas, underscoring the value of task-specific metric prioritization in model selection and suggesting that trade-offs between cost and specific performance metrics are essential in aligning model choice with user needs.

Another worth nothing result is that while Mistral Nemo performed well overall, it was the model that has the most number of hallucinations, which happened especially for larger messages. This model classified a reduced number of messages, namely N=1783, instead of the 1791 samples provided in the dataset. All hallucination cases observed happened when inferencing larger messages.

Finding 4: No model excelled universally in RQ_2. As such, task prioritization and resource constraints should be the primary factors when choosing a model for a specific sentiment analysis application, especially when prompting without examples.

Finally, one trend observed in Section IV-A also continued being true: for most performance metrics, all LLMs evaluated in this work had their zero-shot (without examples) performance metrics overcoming those of the more traditional tools for sentiment analysis utilized in the literature. Those metrics are presented in Table III.

Finding 5: All models had an overall performance that exceeded the more traditional tools utilized in the literature (see Table III).

C. Performance using prompt engineering techniques (RQ_3)

Table IV summarizes the performance metrics (Precision, Recall, and F1-score) for the eight models investigated in this work[9]. The results are group based on the prompt engineering technique employed, namely one-shot, few-shot and chain-of-thought (CoT).

1) One-shot Technique: This technique consists of providing a single example to the LLM as part of the prompt. Since it requires only a single example, it minimizes both the effort of selecting examples and the size of the input that is passed to the model. In our case, since sentiment analysis is a multi-class classification task, we provided three examples, one for each class. Some of our pilot experiment showed that providing a single example from only one class overall made performance metrics worse for the other classes. This was still observed even in more complex techniques such as few-shot and CoT, so we opted to balance the examples per class.

In general, improvements using the One-shot techniques were marginal. These marginal improvements occurred mostly in models that already had good overall performance metrics, such as GPT-4o and mistral-small. For these LLMs, specifically, the example allowed them to achieve more balanced results in terms of precision and recall, improving both of their F1-scores to 0.74. Nevertheless, there were some exceptions, such as the recall for the positive class for LLama3.1 8b (15% improvement). However, this LLM continued to perform badly in other metrics. This was a trend for all the smaller models.

2) Few-shot Technique: This technique consists of providing a few examples (three per class, nine in total for our study) to the large language model as part of the prompt. This builds more contextual understanding before generating predictions.

In summary, for several models, adding a few examples further improved macro and micro recall, precision, and F1-scores. GPT-4o and Mistral Nemo models benefited greatly from few-shot prompting. However, their macro average F1-scores were actually worse than with the one-shot technique. Again, smaller models like Llama3.1 8B did not show significant improvement with few-shot prompts, remaining under 0.6 in F1-scores. This may stem from their more limited memory or attention capacity, which can struggle to synthesize information from multiple examples effectively. On the other hand, few-shot appears to be most beneficial for bigger models that can process and benefit from multiple examples without losing performance. However, even those models did not improve much when compared to one-shot in most cases, showing that there are diminishing returns to providing more examples (however, the quality of examples may affect this).

3) Chain-of-thought (CoT) Technique: In this prompting technique, the model is guided to "think aloud" or generate intermediate reasoning steps before giving a final answer by means of examples that provide this reasoning step. This can help with complex tasks by encouraging step-by-step analysis. We executed this technique with the same nine examples (with reasoning added) utilized in the few-shot technique.

This technique had the most substantial impact on the performance of the models. GPT-4o with CoT reached some of the highest scores across metrics, with a macro F1-score equal to 0.78. This indicates that CoT enables deeper processing and nuanced understanding, critical for sentiment analysis where reasoning can reduce confusion between sentiment categories. The step-by-step reasoning in CoT appears to encourage models to verify and refine their responses, leading to higher precision and recall. For instance, CoT may help a model distinguish subtle linguistic cues signaling negative vs. neutral sentiment more effectively. The biggest example of this happened with the negative class, where most of the bigger models had a large improvement in metrics, especially in recall. GPT-4o's recall in this class improved 26%, allowing its F1-score to improve from 0.62 to 0.78.

Finding 6: Leveraging the chain-of-thought technique allowed GPT-4o to distance itself from the other models and reach the top position in terms of F1-score. However, the difference is not large enough (0.06 in macro avg. F1-score) to justify ignoring previous discussed factors such as cost, flexibility, and privacy.

For smaller models, CoT generally did not result in significant improvements and sometimes led to decreased scores (*e.g.*, Llama3.1 8B stayed near its zero-shot results). One possible reason for this, is that given the reduced memory footprint associated with those LLMs, they may struggle with maintaining coherence in multistep reasoning, resulting in no additional benefit from CoT techniques. In summary, one-shot was most helpful for mid-to-large models. Few-Shot benefited larger models that could process multiple examples, particularly for more ambiguous sentiments. CoT provided the highest boost, particularly in balancing precision and recall, therefore improving F1 scores.

[9]Due to resource limitations, Lllama3.1 70B was only tested for the Chain-of-thought prompting technique.

TABLE V

PERFORMANCE METRICS FOR ALL MODELS TESTED, SEPARATED BY PROMPTING TECHNIQUE APPLIED

Class	Model	One-shot			Few-shot			Chain of Thought (CoT)		
		Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
Positive	GPT 4o	77%	81%	**0.79**	87%	74%	**0.80**	86%	81%	**0.83**
	GPT 4o mini	69%	83%	0.75	72%	79%	0.75	76%	84%	0.80
	Mistral small	**85%**	62%	0.72	**92%**	54%	0.68	**91%**	59%	0.71
	Mistral nemo	79%	79%	**0.79**	80%	76%	0.78	76%	80%	0.78
	Gemma2 27B	70%	78%	0.74	71%	75%	0.73	74%	78%	0.76
	Gemma2 9B	63%	84%	0.72	65%	82%	0.73	63%	86%	0.72
	Llama3.1 70B	-	-	-	-	-	-	69%	87%	0.77
	Llama3.1 8B	47%	**93%**	0.63	50%	**91%**	**0.64**	48%	**93%**	0.63
Negative	GPT 4o	69%	**62%**	0.65	**78%**	49%	0.60	78%	**66%**	**0.71**
	GPT 4o mini	72%	45%	0.55	74%	43%	0.54	**79%**	48%	0.60
	Mistral small	**74%**	53%	0.62	75%	47%	0.58	75%	58%	0.65
	Mistral nemo	72%	**62%**	0.67	76%	48%	0.59	68%	**66%**	0.67
	Gemma2 27B	71%	59%	0.65	70%	**56%**	0.62	70%	65%	0.67
	Gemma2 9B	65%	56%	0.60	66%	53%	0.59	67%	51%	0.58
	Llama3.1 70B	-	-	-	-	-	-	71%	53%	0.60
	Llama3.1 8B	64%	52%	0.58	65%	53%	0.58	62%	56%	0.59
Neutral	GPT 4o	**76%**	77%	**0.76**	69%	89%	**0.78**	**75%**	84%	**0.80**
	GPT 4o mini	69%	74%	0.71	67%	78%	0.72	70%	80%	0.75
	Mistral small	65%	85%	0.74	63%	**91%**	0.74	66%	**88%**	0.75
	Mistral nemo	73%	**78%**	**0.76**	69%	84%	0.75	74%	73%	0.74
	Gemma2 27B	71%	72%	0.72	69%	74%	0.71	74%	74%	0.74
	Gemma2 9B	70%	60%	0.65	69%	65%	0.67	70%	62%	0.66
	Llama3.1 70B	-	-	-	-	-	-	73%	71%	0.72
	Llama3.1 8B	73%	36%	0.48	**72%**	42%	0.53	**79%**	36%	0.49
Macro Avg.	GPT 4o	74%	**74%**	**0.74**	**78%**	**71%**	**0.73**	**80%**	**77%**	**0.78**
	GPT 4o mini	70%	67%	0.67	71%	67%	0.67	75%	71%	0.72
	Mistral small	**75%**	67%	0.69	77%	64%	0.67	77%	68%	0.71
	Mistral nemo	**75%**	73%	**0.74**	75%	69%	0.71	73%	73%	0.73
	Gemma2 27B	71%	70%	0.70	70%	68%	0.69	73%	72%	0.72
	Gemma2 9B	66%	67%	0.65	67%	67%	0.66	66%	66%	0.65
	Llama3.1 70B	-	-	-	-	-	-	71%	70%	0.70
	Llama3.1 8B	61%	60%	0.56	62%	62%	0.58	63%	62%	0.57
Micro Avg.	GPT 4o	**75%**	**75%**	**0.75**	**75%**	**75%**	**0.75**	**79%**	**79%**	**0.79**
	GPT 4o mini	69%	69%	0.69	70%	70%	0.70	73%	73%	0.73
	Mistral small	71%	71%	0.71	69%	69%	0.69	72%	72%	0.72
	Mistral nemo	**75%**	**75%**	**0.75**	73%	73%	0.73	73%	73%	0.73
	Gemma2 27B	71%	71%	0.71	70%	70%	0.70	73%	73%	0.73
	Gemma2 9B	66%	66%	0.66	67%	67%	0.67	66%	66%	0.66
	Llama3.1 70B	-	-	-	-	-	-	71%	71%	0.71
	Llama3.1 8B	57%	57%	0.57	59%	59%	0.59	57%	57%	0.57

Finding 7: As more complex prompt engineering techniques are employed, the boost to bigger models became larger. Conversely, fewer benefits could be observed for smaller models, and, in some cases, they became worse than zero-shot.

V. THREATS TO VALIDITY

We discuss threats to the study validity in what follows. We adopted the taxonomy proposed by Wohlin et al. [49].

Construct and Internal Validity. We only experimented a limited sample of LLMs in this study. This means that other LLMs (or different variations of the models chosen) can perform better (or worse) than the results obtained in this study. Newer and/or more powerful LLMs may also have been released at the time this study is published. We also only experimented with a limited number of prompting techniques, which means that our results might not represent the best possible scenario for each of the models.

The data utilized in this work was manually labeled by humans. The steps taken to create the dataset might introduce bias to this work, especially since we only utilized a single dataset. However, we selected a dataset that has taken several measures to reduce bias (*e.g.*, multiple experts validated each label, a framework from literature was utilized to guide the labeling process, etc.). The selection of examples and rationales for classification that were utilized in this study might be subject to bias. To mitigate this, two of the authors worked in a pair to select and review the examples.

Finally, due to hardware constraints, for the models that were locally executed via Ollama, we executed quantized versions of the models. This is a technique that reduces the hardware requirements of running a model, by quantizing the weights, but that introduces tradeoffs on quality. We mitigated that by utilizing the default quantization that the Ollama tool suggests, Q4_0, which they describe as a good balance between performance and scalability. The results observed in this study could potentially be different if the full version of the models is executed instead of the quantized versions.

Conclusion and External Validity. As aforementioned, our results were obtained by utilizing only one dataset. This means that our results are only representative of a limited number of projects (*i.e.*, 36 software projects) and a specific context (*i.e.*, pull request discussions). As such, our results cannot generalize the performance of LLMs when performing sentiment analysis in other contexts.

VI. Implications and Conclusion

This study evaluates the efficacy of eight large language models (LLMs) in performing sentiment analysis on GitHub pull request discussions, specifically examining each model's performance across different metrics and considering the effects of three prompt engineering techniques. Our research highlights several key implications and contributions for developers, tool builders, and researchers, which are as follows.

Traditional Tools vs. LLMs: Our results revealed that, on most metrics, all large language models experimented outperform traditional, the domain-specific, traditional sentiment analysis tools commonly used in software engineering. These improvements in performance metrics suggest that LLMs can better handle the contextual and technical nuances of software discussions, demonstrating their suitability for applications that involve complex sentiment interpretation within collaborative development platforms.

No model is universally better: While GPT-4o achieved the best overall performance, no single model excelled across all tasks or met every requirement for all tasks. Each model demonstrated unique strengths across different classes (positive, negative, neutral), allowing for targeted model selection based on task-specific needs, such as toxicity detection or sentiment-based assignment prioritization. This outcome emphasizes the need for developers to assess specific task requirements and resource constraints when selecting a model for sentiment analysis in software engineering.

Part of the LLM selection process also should consider privacy and scalability concerns. Open-source models demonstrated competitive performance and offer advantages in cost and flexibility. Their deployability in local environments supports data privacy, essential for organizations handling sensitive data. However, proprietary models like GPT-4o (our overall best performer) provide ease of deployment via cloud APIs and lower hardware requirements, making them suitable for applications where data privacy is not the primary concern (e.g., when there is a preprocessing step to anonymize the data). Nevertheless, depending on the resource availability and task requirements, it might be advantageous to host open-source models in the cloud.

Prompt-engineering techniques are more effective on larger models: We found that prompt-engineering techniques, particularly the CoT approach, notably improved performance metrics, especially for larger models like GPT-4o. While these techniques enhance the reasoning process and context comprehension, smaller models saw limited benefit, suggesting that prompt complexity should align with model capacity. This insight is particularly valuable for practitioners seeking to optimize model efficiency and cost.

This paper provides a comprehensive evaluation of LLM applicability in sentiment analysis tasks within the context of GitHub pull request discussions. Our findings indicate that while LLMs, particularly with prompt engineering, offer substantial improvements over traditional tools, choosing an optimal model depends on prioritizing performance metrics that align with specific task goals. By showcasing these findings, this work serves as a valuable reference for building scalable, task-specific sentiment analysis tools, contributing to the enhancement of collaborative and feedback-driven software development environments.

Data availability statement

All scripts and data utilized in this study are available in the replication package [47].

Acknowledgements

Omitted for blind review.

References

[1] J. Tantisuwankul, Y. S. Nugroho, R. G. Kula, H. Hata, A. Rungsawang, P. Leelaprute, and K. Matsumoto, "A topological analysis of communication channels for knowledge sharing in contemporary github projects," *Journal of Systems and Software*, vol. 158, p. 110416, 2019.

[2] E. Kalliamvakou, D. Damian, K. Blincoe, L. Singer, and D. M. German, "Open source-style collaborative development practices in commercial projects using github," in *2015 IEEE/ACM 37th IEEE international conference on software engineering*, vol. 1. IEEE, 2015, pp. 574–585.

[3] M. M. Rahman and C. K. Roy, "An insight into the pull requests of github," in *11th working conference on mining software repositories*, 2014, pp. 364–367.

[4] X. Zhang, Y. Yu, G. Gousios, and A. Rastogi, "Pull request decisions explained: An empirical overview," *IEEE Transactions on Software Engineering*, vol. 49, no. 2, pp. 849–871, 2022.

[5] U. A. Mannan, I. Ahmed, C. Jensen, and A. Sarma, "On the relationship between design discussions and design quality: A case study of apache projects," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 543–553.

[6] X. Zhang, Y. Yu, G. Gousios, and A. Rastogi, "Pull request decisions explained: An empirical overview," *IEEE Transactions on Software Engineering*, vol. 49, no. 2, pp. 1–15, 2023.

[7] C. Barbosa, A. Uchôa, D. Coutinho, W. K. Assunçao, A. Oliveira, A. Garcia, B. Fonseca, M. Rabelo, J. E. Coelho, E. Carvalho *et al.*, "Beyond the code: Investigating the effects of pull request conversations on design decay," in *2023 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 2023, pp. 1–12.

[8] J. Tsay, L. Dabbish, and J. Herbsleb, "Influence of social and technical factors for evaluating contribution in github," in *Proceedings of the 36th international conference on Software engineering*, 2014, pp. 356–366.

[9] M. El Mezouar, F. Zhang, and Y. Zou, "An empirical study on the teams structures in social coding using github projects," *Empirical Software Engineering*, vol. 24, no. 6, pp. 3790–3823, 2019.

[10] E. Guzman and B. Bruegge, "Towards emotional awareness in software development teams," in *Proceedings of the 2013 9th joint meeting on foundations of software engineering*, 2013, pp. 671–674.

[11] N. C. Dang, M. N. Moreno-García, and F. De la Prieta, "Sentiment analysis based on deep learning: A comparative study," *Electronics*, vol. 9, no. 3, p. 483, 2020.

[12] E. Guzman, D. Azócar, and Y. Li, "Sentiment analysis of commit comments in github: An empirical study," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 352–355.

[13] F. Calefato, F. Lanubile, and N. Novielli, "Sentiment polarity in collaborative software development," in *Proceedings of the 40th International Conference on Software Engineering*. IEEE, 2018, pp. 570–580.

[14] N. Novielli, D. Girardi, and F. Lanubile, "A benchmark study on sentiment analysis for software engineering research," in *Proceedings of the 15th International Conference on Mining Software Repositories*, 2018, pp. 364–375.

[15] M. R. Islam and M. F. Zibran, "Leveraging automated sentiment analysis in software engineering: Existing techniques and open challenges," in *Proceedings of the 15th International Conference on Mining Software Repositories*. ACM, 2018, pp. 544–547.

[16] F. Calefato, F. Lanubile, and N. Novielli, "Emotxt: A toolkit for emotion recognition from text," in *Proceedings of the Seventh International Conference on Affective Computing and Intelligent Interaction*. IEEE, 2017, pp. 79–85.

[17] B. Lin and B. Vasilescu, "Can we use se-specific sentiment analysis tools in pull request discussions? an empirical study," in *Proceedings of the 15th International Conference on Mining Software Repositories*. ACM, 2018, pp. 368–371.

[18] F. Calefato, F. Lanubile, and N. Novielli, "How far are we from sentiment analysis for pull requests? a critical evaluation of se-specific tools," *Empirical Software Engineering*, vol. 24, no. 2, pp. 851–879, 2019.

[19] T. B. Brown, B. Mann, N. Ryder *et al.*, "Language models are few-shot learners," 2020.

[20] S. Gururangan, A. Marasovic, S. Swayamdipta, K. Lo, I. Beltagy, D. Downey, and N. A. Smith, "Don't stop pretraining: Adapt language models to domains and tasks," *arXiv preprint arXiv:2004.10964*, 2020.

[21] D. Coutinho, L. Cito, M. V. Lima, B. Arantes, J. A. Pereira, J. Arriel, J. Godinho, V. Martins, P. Libório, L. Leite, A. Garcia, W. K. G. Assunção, I. Steinmacher, A. Baffa, and B. Fonseca, ""looks good to me ;-)": Assessing sentiment analysis tools for pull request discussions," in *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering (EASE 2024)*. Salerno, Italy: ACM, 2024, p. 11.

[22] M. A. Hasan, S. Das, A. Anjum, F. Alam, A. Anjum, A. Sarker, and S. R. H. Noori, "Zero- and few-shot prompting with llms: A comparative study with fine-tuned models for bangla sentiment analysis," *arXiv preprint arXiv:2308.10783v2*, 2024. [Online]. Available: https://arxiv.org/pdf/2308.10783v2.pdf

[23] B. Pang, L. Lee *et al.*, "Opinion mining and sentiment analysis," *Foundations and Trends® in information retrieval*, vol. 2, no. 1–2, pp. 1–135, 2008.

[24] B. Liu, *Sentiment analysis and opinion mining*. Springer Nature, 2022.

[25] D. Graziotin, X. Wang, and P. Abrahamsson, "Happy software developers solve problems better: psychological measurements in empirical software engineering," *PeerJ*, vol. 2, p. e289, 2014.

[26] ——, "How do you feel, developer? an explanatory theory of the impact of affects on programming performance," *PeerJ Computer Science*, vol. 1, p. e18, 2015.

[27] M. Herrmann and J. Klünder, "From textual to verbal communication: Towards applying sentiment analysis to a software project meeting," in *Leibniz University Hannover*, 2021.

[28] Q. T. Ain, M. Ali, A. Riaz, A. Noureen, M. Kamran, B. Hayat, and A. Rehman, "Sentiment analysis using deep learning techniques: a review," *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 6, 2017.

[29] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[30] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[31] T. Zhan, C. Shi, Y. Shi, H. Li, and Y. Lin, "Optimization techniques for sentiment analysis based on llm (gpt-3)," *arXiv preprint arXiv:2405.09770*, 2024.

[32] J. Niimi, "Dynamic sentiment analysis with local large language models using majority voting: A study on factors affecting restaurant evaluation," *arXiv preprint arXiv:2407.13069*, 2024.

[33] F. Xing, "Designing heterogeneous llm agents for financial sentiment analysis," *arXiv preprint arXiv:2401.05799*, 2024.

[34] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou, "Chain of thought prompting elicits reasoning in large language models," *arXiv preprint arXiv:2201.11903*, 2022.

[35] Y. Zhou, A. I. Muresanu, Z. Han, K. Paster, S. Pitis, H. Chan, and J. Ba, "Large language models are human-level prompt engineers," *arXiv preprint arXiv:2211.01910*, 2022.

[36] M. Thelwall, K. Buckley, and G. Paltoglou, "Sentiment strength detection for the social web," *Journal of the American Society for Information Science and Technology*, vol. 63, no. 1, pp. 163–173, 2012.

[37] O. Kolchyna, T. T. Souza, P. Treleaven, and T. Aste, "Twitter sentiment analysis: Lexicon method, machine learning method and their combination," *arXiv preprint arXiv:1507.00955*, 2015.

[38] M. R. Islam and M. F. Zibran, "Sentistrength-se: Exploiting domain specificity for improved sentiment analysis in software engineering text," *Journal of Systems and Software*, vol. 145, pp. 125–146, 2018.

[39] T. Zhang, I. C. Irsan, and F. Thung, "Revisiting sentiment analysis for software engineering in the era of large language models," 2023. [Online]. Available: https://arxiv.org/abs/2310.11113

[40] "An empirical evaluation of the zero-shot, few-shot, and traditional fine-tuning based pretrained language models for sentiment analysis in software engineering," 2023.

[41] N. Novielli, F. Calefato, D. Dongiovanni, D. Girardi, and F. Lanubile, "Can we use se-specific sentiment analysis tools in a cross-platform setting?" in *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020, pp. 158–168.

[42] M. Obaidi, L. Nagel, A. Specht, and J. Klünder, "Sentiment analysis tools in software engineering: A systematic mapping study," *Information and Software Technology*, p. 107018, 2022.

[43] J. Tsay, L. Dabbish, and J. Herbsleb, "Let's talk about it: Evaluating contributions through discussion in github," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 144–154. [Online]. Available: https://doi.org/10.1145/2635868.2635882

[44] G. Viviani, M. Famelis, X. Xia, C. Janik-Jones, and G. C. Murphy, "Locating latent design information in developer discussions: A study on pull requests," *IEEE Transactions on Software Engineering*, vol. 47, no. 7, pp. 1402–1413, 2019.

[45] M. Ortu, A. Murgia, G. Destefanis, P. Tourani, R. Tonelli, M. Marchesi, and B. Adams, "The emotional side of software developers in jira," in *Proceedings of the 13th International Conference on Mining Software Repositories*, ser. MSR '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 480–483. [Online]. Available: https://doi.org/10.1145/2901739.2903505

[46] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, "Large language models for software engineering: A systematic literature review," *ACM Transactions on Software Engineering and Methodology*, 2023.

[47] "Replication package," https://anonymous.4open.science/r/LLM4SA-replication-C2D9/.

[48] G. Marvin, N. Hellen, D. Jjingo, and J. Nakatumba-Nabende, "Prompt engineering in large language models," in *International conference on data intelligence and cognitive informatics*. Springer, 2023, pp. 387–402.

[49] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*, 1st ed. Springer Science & Business Media, 2012.