Shedding Light on Software Engineering-specific Metaphors and Idioms

Mia Mohammad Imran
Virginia Commonwealth University
Richmond, Virginia, USA
imranm3@vcu.edu

Preetha Chatterjee
Drexel University
Philadelphia, Pennsylvania, USA
preetha.chatterjee@drexel.edu

Kostadin Damevski
Virginia Commonwealth University
Richmond, Virginia, USA
kdamevski@vcu.edu

ABSTRACT

Use of figurative language, such as metaphors and idioms, is common in our daily-life communications, and it can also be found in Software Engineering (SE) channels, such as comments on GitHub. Automatically interpreting figurative language is a challenging task, even with modern Large Language Models (LLMs), as it often involves subtle nuances. This is particularly true in the SE domain, where figurative language is frequently used to convey technical concepts, often bearing developer affect (e.g., *'spaghetti code'*). Surprisingly, there is a lack of studies on how figurative language in SE communications impacts the performance of automatic tools that focus on understanding developer communications, e.g., bug prioritization, incivility detection. Furthermore, it is an open question to what extent state-of-the-art LLMs interpret figurative expressions in domain-specific communication such as software engineering. To address this gap, we study the prevalence and impact of figurative language in SE communication channels. This study contributes to understanding the role of figurative language in SE, the potential of LLMs in interpreting them, and its impact on automated SE communication analysis. Our results demonstrate the effectiveness of fine-tuning LLMs with figurative language in SE and its potential impact on automated tasks that involve affect. We found that, among three state-of-the-art LLMs, the best improved fine-tuned versions have an average improvement of 6.66% on a GitHub emotion classification dataset, 7.07% on a GitHub incivility classification dataset, and 3.71% on a Bugzilla bug report prioritization dataset.

ACM Reference Format:
Mia Mohammad Imran, Preetha Chatterjee, and Kostadin Damevski. 2024. Shedding Light on Software Engineering-specific Metaphors and Idioms. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3597503.3639585

1 INTRODUCTION

Figurative language is the use of words or phrases in a way that deviates from their literal meaning, aiming to evoke specific concepts or imagery within one's imagination[1]. Figurative language consists of different types [1], such as metaphors, which use comparisons to describe something differently (e.g., *"the road ahead is a long and winding journey"*); idioms, which are common phrases that have alternate meanings (e.g., *'to beat around the bush'*); similes, which use *'like'* or *'as'* to compare two things (e.g., *'as light as a feather'*); and personification, which gives human qualities to objects or animals (e.g., *'the leaves danced in the wind'*).

Within the software engineering (SE) community, professionals often employ various distinctive figurative expressions that are not commonly used in everyday discourse. For instance, developers utilize the metaphorical term *'anti-pattern'* to communicate the idea of a recurring problem that should be avoided [2]. Idioms, another frequently employed form of figurative expression, play a crucial role in SE communication by succinctly and colloquially conveying common ideas or concepts. An example of this is when developers describe poorly written code as *'spaghetti code'*, implying that it is convoluted and challenging to comprehend [3].

Just as humans use phrases like *'boil'* with anger or *'a breath of fresh air'* for relief [4], developers might say *'a thorn in my side'*[2] to express persistent annoyance or difficulty with an API or a feature. Use of pejorative terms like *'garbage code'* [3], *'Frankencode'* [4] can be indicators of severe negative emotions, leading to toxic discussions. Therefore, understanding the use of figurative language in software development discourse can help detect the use of offensive language [5] and provide valuable insights into the overall health of a software project [6]. Developers also use figurative expressions to indicate the impact and severity of a bug. For instance, while expressions like *'a ticking time bomb'* [5], suggests significant future problems, *'showstopper'* [6] and *'critical roadblock'* [7], emphasize the urgency of addressing the bug at the present. A recent blog article noted how an improved understanding of software engineering metaphors would mitigate the risks of misinterpretations, fostering more precise and effective communication within the software development community[8], while a recent study showed figurative language like Humor has positive effect on developer engagement [7]. Recent research studies have also highlighted that flaws in SE emotion and sentiment detection tools stem from the use of figurative language [8–11]. On a Stack Overflow and GitHub dataset, Novielli et al. [8] found that 9% of the errors in sentiment analysis were due to figurative language, noting that it poses an open challenge for sentiment detection in software engineering.

[1] https://www.ldoceonline.com/dictionary/figurative

ICSE '24, April 14–20, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0217-4/24/04.
https://doi.org/10.1145/3597503.3639585

[2] https://github.com/chipsalliance/chisel/pull/3352#issuecomment-1593479230
[3] https://github.com/adamit24/countdownClass/issues/1
[4] https://github.com/drupal-code-builder/drupal-code-builder/issues/165
[5] https://github.com/godotengine/godot/issues/77480#issue-1726201628
[6] https://github.com/conwid/VSCleanBin/issues/2#issuecomment-571094076
[7] https://github.com/canjs/canjs/pull/3286
[8] https://www.leadingagile.com/2018/11/metaphorically-speakingthe-history-of-communication-in-software/

Despite its potential impact on the performance of automatic tools focused on understanding SE text, there have been very limited studies on analyzing figurative language in SE, e.g., there have been some studies on SE synonyms [12, 13] and programming language-specific idioms [14]. In this paper, we aim to go beyond the synonyms and explore the broader landscape of figurative language in SE. We aim to *'shed light on'* or analyze the use of figurative language (specifically, metaphors and idioms) in SE communication channels and contribute to the understanding of how recently proposed language models that target software-related text can be made to recognize figurative expressions.

Large Language Models (LLMs), such as BERT [15] and RoBERTa [16], have recently demonstrated state-of-the-art results on a variety of software engineering tasks, e.g., code completion, code review, bug localization, sentiment analysis, toxicity detection [17–22]. While LLMs are not explicitly designed to detect figurative languages like metaphors and idioms, they can acquire this ability through training on large datasets such as Wikipedia and Stack Overflow [23–26]. This capability is particularly beneficial in the software engineering context, as it enables a more nuanced and accurate analysis of developer communications. Without this ability, an LLM may misinterpret or misclassify text, leading to erroneous results. For instance, if an LLM cannot recognize the idiom *'edge case'*, it may interpret the phrase literally and erroneously categorize the text as being related to a specific type of physical boundary instead of grasping its figurative meaning of a rare or unusual scenario.

Through this study, we will examine the relevance of figurative language in GitHub communication channels, the ability of LLMs to detect figurative language in the SE context, and the impact of figurative language on affect analysis and bug report priority detection. By gaining a deeper understanding of the role and effects of figurative language in SE, we aim to contribute to the development of more effective and accurate NLP-based systems for SE tasks, specifically in automated recognition of developer emotions and incivility on GitHub, and bug report priority detection. We focus on answering the following three research questions:

RQ1: How well can existing LLMs interpret figurative language (i.e., metaphors and idioms) used in software engineering?
To answer this RQ, we collect a set of 2000 sentences containing figurative language and create *rephrased* sentences, i.e., sentences with similar meanings but without figurative expressions. We also create *altered* sentences that share as many words as the original sentence but convey different meanings, e.g., using metaphors in their literal sense or using idioms in a different context other than software engineering. This procedure of creating, so called, entailed and non-entailed text from premise text has been widely used in NLP [27–29]. Using this data triple of original, rephrased, and altered meaning sentences, we investigate whether LLMs can recognize the semantics of figurative sentences by computing how often the models identified the semantic dissimilarity of the rephrased sentence with the altered sentence. Our results suggest that LLMs have a limited ability to interpret figurative language, with higher performance for general figurative expressions than software engineering-specific ones.

RQ2: Can the performance of software engineering-specific affective analysis be improved by a better insight into figurative language?
Affect expressions are the means to convey emotions, feelings, and attitudes to others [30]. For some time now, researchers have been exploring automatic affect analysis, which encompasses tasks such as emotion analysis, sentiment analysis, and incivility analysis. To answer RQ2, we fine-tune several LLMs using contrastive learning [31] with our dataset of figurative language in order to improve their ability to interpret figurative language. We then compare the performance of the fine-tuned LLMs to the original models of two publicly available affect datasets: an emotion dataset curated from GitHub, and an incivility dataset curated from GitHub. Our results indicate that fine-tuned LLMs perform better in both cases.

RQ3: Can a better understanding of figurative language enhance software engineering automation where affect plays a role?
A number of research tasks in SE indirectly involve affective natural language text, e.g., app review analysis, opinion mining [32, 33]. Specifically, in this RQ we investigate how a better understanding of figurative language can impact bug report priority detection, which is a significant area of interest in open-source software research [34–37]. Umer et al. observed that emotions influence bug report priority detection [36]. To address this problem, recently researchers have employed Language Models (LLMs) [34]. In this study, we explore LLMs fine-tuned with contrastive learning using our figurative language dataset, similar to the approach in RQ2, and conducted experiments on the publicly available Bugzilla dataset[9]. Our results indicate that fine-tuning with our figurative language dataset improves bug report priority detection.

We publish the annotation instructions, annotated dataset, and source code to facilitate the replication of our study at https://github.com/vcu-swim-lab/SE-Figurative-Language.

2 DATASET

To conduct our study, we curate a dataset of developer communications containing figurative language. Towards that goal, we first collect data from GitHub issues and pull requests and identify the occurrences of idioms and metaphors. To inquire whether language models understand figurative language, we manually rephrase the original sentences containing figurative language to generate: 1) sentences that are similar in meaning to the original but do not contain idioms or metaphors; and 2) sentences that contain similar words as the original sentences but are semantically dissimilar, i.e., have a different meaning. In this section, we detail each step involved in constructing our dataset.

2.1 Data Collection

We selected nine popular GitHub repositories, each with a minimum of 50k stars: skylot/jadx, laravel/laravel, microsoft/PowerToys, rails/rails, redis/redis, facebook/react, tensorflow/tensorflow, huggingface/transformers, and microsoft/vscode. We collected 10k comments from each repository (5k PR comments and 5k issue comments) between February 2022 and May 2023. We split the comments into sentences using NLTK[10] and filtered out sentences with fewer than 5 words, resulting in a total of 202k sentences.

[9] https://bugs.eclipse.org/bugs/
[10] https://www.nltk.org/

One of our study's end goals is to examine figurative language's impact on affective expressions in a software engineering context. Previous research has shown that most comments on GitHub are neutral, lacking any detectable emotions or sentiments [38]. Therefore, we excluded neutral sentences by using a software engineering-specific sentiment analysis tool [39].

In addition, to avoid including sentences that do not contain any figurative expressions, we applied a popular metaphor detection [23] and an idiom detection tool [24] to identify candidate metaphors and idioms in each sentence. This model-in-the-loop approach is popular in Natural Language Inference (NLI) research, e.g., figurative language interpretation [40, 41], as it maximizes the value of annotation effort, which requires tedious human labor. We discarded sentences that do not contain any candidate idioms or metaphors. We randomly selected 1000 sentences containing metaphors from the remaining sentences. We also randomly chose 1000 sentences containing idioms (different from the metaphor set). This process resulted in a dataset of 2000 sentences.

2.2 Data Annotation

First, we recruited four annotators (two graduate students and two senior undergraduate students) who were each given 500 sentences to annotate (250 with metaphors and 250 with idioms). Due to the nature of the task and difficulties with crowd-sourcing [29], we opted for a small number of annotators that are native speakers/professionally fluent in English with a strong computer science background. Along with the 2000 sentences in total, the annotators were provided with a set of candidate figurative expressions marked by the above-mentioned tools. We instructed them to: 1) verify the candidates as metaphors or idioms and judge whether each metaphor or idiom is specific to software engineering or general purpose; and 2) create rephrased sentences from the original. We also held a short training session in which we reviewed the annotation process for a few representative examples with each annotator. Below, we describe these data annotation steps in detail (see also Figure 1).

2.2.1 Verifying Figurative Expressions. For verifying metaphors and idioms, we followed best practices from existing literature. More specifically, to verify the metaphors we asked the annotators to carefully read the Metaphor Identification Procedure (MIP) guideline by the Pragglejaz Group [42]. The MIP guideline is a well-known procedure for identifying metaphors. Based on the guideline, the annotators marked the correct metaphoric expressions from the candidate set. For example, the annotators confirmed that *'nasty bug'* is a valid metaphor for a difficult fault in the sentence, *"Otherwise, this could give us a nasty bug."*

We noted in the annotation instructions that most metaphors are conventional, i.e., metaphors that are often used in everyday language [43]. For example, in the following sentence: *"I see your point"*, *'see'* and *'point'* both are metaphors [43]. Often such cases can be observed in software engineering communication. For instance, *'pinging'* in the following sentence is a metaphor: *"Hi @[USER], thanks for pinging me on this issue."* Here, *'pinging'* is a colloquial way of saying *'contacting someone'*, while the literal meaning of *'pinging'* comes from computer networking terminology[11].

For verifying idioms, we followed the guideline provided by Stowe et al. [27], which asked the annotators to look up idioms in popular dictionaries (such as the Oxford English Dictionary[12], the Webster Dictionary[13], and the Longman Dictionary of Contemporary English[14] and popular search engines (e.g., Google). We instructed the annotators to consider an expression as likely to be an idiom if its dictionary definition is: 1) applicable in the context; and 2) a good syntactic fit in the same environment. For example, in the sentence, *"I will also be keeping an eye on you"*, *'keeping an eye'* is an idiom which means *'to watch someone or something or stay informed about the person's behavior, especially to keep someone out of trouble.'*[15] Conversely, when the meaning of the candidate idiomatic expression is literal in the context of the sentence and the dictionary definition is not applicable, it is likely not to be an idiom. For example, in the sentence, *"It was cold, so cold in the jeep that it was with difficulty that Alexei kept his eyes open"*, *'kept his eyes open'* is not an idiom. Since software-specific words have distinct meanings from conventional terms (e.g., bug, issue, error, function), we supplied annotators with established software engineering glossary terms from the FDA[16] and Google[17].

Once annotators verified the candidate set, we asked them to mark whether the figurative expressions were software engineering-specific or general-purpose. The annotators identified 752 sentences with metaphors and 909 with idioms, totaling 1661 sentences. The remaining 339 sentences did not contain any figurative words. These 1661 sentences had a total of 1741 unique figurative expressions, with 445 being SE-specific and 1296 general.

2.2.2 Rephrasing Sentences. The process of rephrasing sentences was divided into two phases: creating semantically-equivalent rephrased sentences and constructing altered-meaning sentences. We refer to the semantically equivalent rephrased sentences as Equivalent Meaning Sentence (EMS) throughout the paper. These sentences retain the original meaning of the sentence, but the figurative expressions are replaced with literal terms. We refer to the altered-meaning sentences as Different Meaning Sentence (DMS). These sentences are modified so that they significantly differ in meaning from the original sentences.

a) EMS Construction: The annotators were tasked with rephrasing each sentence on their list, i.e., removing the (verified) figurative expressions while maintaining the original semantics of the sentence as much as possible. In other words, the replaced figurative expression should entail its literal counterpart. For example, in the sentence, *"[USER] Thanks for your help, what you said may be a hidden bug."*, the figurative expression *'hidden bug'* is replaced with *'unseen error'* resulting in the EMS: *"[USER] Thanks for your help, what you said may be an unseen error."* This approach is inspired by

[11] https://ftp.arl.army.mil/~mike/ping.html
[12] https://www.oxfordlearnersdictionaries.com/
[13] https://www.merriam-webster.com/
[14] https://www.ldoceonline.com/
[15] https://dictionary.cambridge.org/
[16] https://www.fda.gov/inspections-compliance-enforcement-and-criminal-investigations/inspection-guides/glossary-computer-system-software-development-terminology-895
[17] https://developers.google.com/machine-learning/glossary

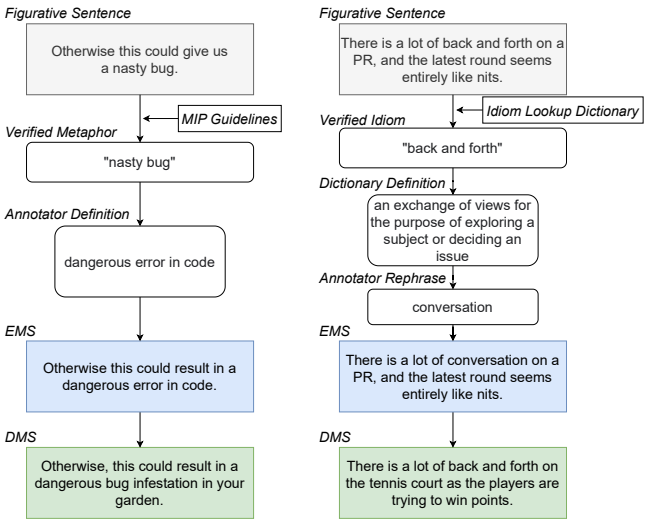


Figure 1: Figurative language annotation procedure.

previous research by Stowe et al. on figurative language in NLP [27]. It is worth noting that for EMS we did not employ multiple annotators to annotate the same set or calculate inter-annotator agreement as Stowe et al. found that this method does not yield significantly different quality compared to the conventional approach [27].

b) DMS Construction: Different Meaning Sentences are variations of metaphorical or idiomatic sentences that convey a different meaning than the original sentence and do not entail it [27]. Two strategies were employed to construct DMS: a) using figurative expressions in a literal sense; and b) replacing the figurative expressions and their context with different words. These strategies are inspired by previous research on figurative language in natural language processing [27, 44, 45]. We also apply a model-in-the-loop approach for DMS generation [40, 41]. More specifically, we first generated four candidate DMSs for each sentence, two each for each strategy using GPT-4 [46] API ('gpt-4'[18]), and, second, we recruited human annotators to select the best-generated candidate (or to create one of their own if none is available).

Candidate DMS Generation. ChatGPT [46] has shown promising results in data annotation tasks, including text generation, in some cases outperforming human crowd-workers [47, 48]. Following recent literature, we create two GPT-4 prompts for the two different strategies for DMS generation [48, 49]. The prompts were carefully devised by using the existing literature on this topic [27, 44, 45]. The prompts for generating DMS are as follows:

Generating DMS by using the figurative language in a literal manner:

> You are reading GitHub comments with figurative expressions. Your task is to generate 2 examples by using the given figurative expressions in a literal manner to construct different sentences. Do not replace them. Add/change new contexts if necessary. The new sentence must have a completely different meaning than the original. You must keep the semantic order of the original sentences as much as possible. Don't explain your answer.
>
> Original Sentence:*<insert utterance>*.
>
> Figurative expressions: *<insert figurative expressions>*

Generating DMS by replacing the figurative language:

> You are reading GitHub comments with figurative expressions. Your task is to generate 2 examples by replacing given figurative expressions to construct different sentences. The new sentence must have a completely different meaning than the original. You are only allowed to change the figurative expression and its context. You must keep the semantic order of the original sentences as much as possible. Don't explain your answer.
>
> Original Sentence:*<insert utterance>*.
>
> Figurative expressions: *<insert figuration expressions>*

DMS Selection. Two additional annotators (one of the authors and one senior undergraduate student) were responsible for the candidate selection of the DMS. We provided the annotators the original sentence, the figurative expressions, and the list of candidate DMSs with the following instructions: *"You will be provided with 4 candidate sentences, two of which come from Type 1 and two come from Type 2. Choose the best 1 out of the 4 candidates, with a preference towards choosing from Type 1. If none of these 4 are good candidates, write None. When choosing, try to choose a sentence that has 1) similar semantic order to the original sentence, and 2) a different meaning than the original sentence."*

We instructed the annotators to write their own DMS when their selection is *'None'*. Once they completed an annotation pass over the entire dataset, the two annotators met in person in order to discuss the 310 cases where they disagreed (i.e., selected different DMS candidates or *'None'*) and resolved them in order to achieve 100% agreement. This human-in-the-loop methodology helps with the more difficult task of DMS generation, enhancing the overall quality and efficiency the process. The iterative resolution of differences ensured a high quality of annotated data.

3 PREVALENCE OF SE-SPECIFIC FIGURATIVE LANGUAGE

In order to understand if SE-specific figurative language appears frequently in the wild, we examine the frequency of occurrence of figurative language in a large sample of developer communication on GitHub. More specifically, we collected 1,000 issue comments and 1,000 pull request comments for each of the top 100 repositories by star count on GitHub, i.e., a total of 200k comments. We analyzed comments made from September 1, 2022, to January 1, 2023, spanning 4 months, and excluded repositories with fewer

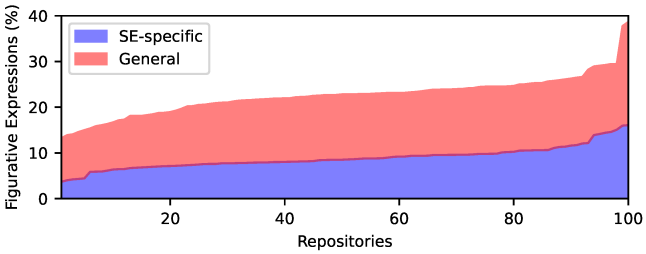
[18] https://platform.openai.com/docs/models/gpt-4-and-gpt-4-turbo/

than 1,000 issue and pull request comments during this time. The collected 200k comments were split into a total of 484k sentences using NLTK[19]. Leveraging our annotated dataset consisting of 1741 unique figurative expressions (445 SE-specific and 1296 general), we searched for matches in the set of 484k sentences after applying standard NLP pre-processing (removing punctuation and non-alphabet characters, and lemmatization using SpaCy) since some of the figurative expressions have different spelling variations (e.g., *'root cause'*, *'root-cause'*, and *'root causes'*). To ensure that the matches were not spuriously identifying figurative language (due to polysemy), we also executed the metaphor and idiom detection tools [23, 24], the same ones that we use for candidates generation, selecting only the matches that were also confirmed by one of these tools.

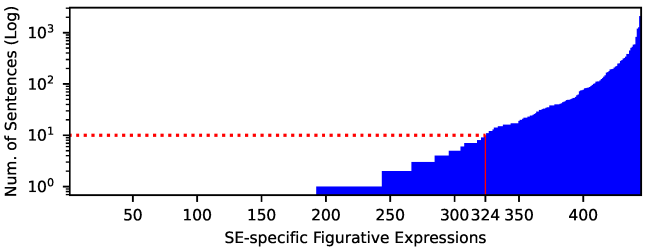
Of the examined 484k sentences, the 445 SE-specific figurative expressions that annotators identified occurred in 44k sentences (9%), while the 1296 general figurative expressions occurred in 107k sentences (22%). Some sentences (2.67%) had both SE-specific and general figurative expressions.

The distribution of general and SE-specific figurative expressions in different GitHub repositories is shown in Figure 2a. SE-specific figurative language does appear in non-trivial amounts in most repositories we examined (i.e., in between 3.69% and 16.08% of sentences), but much less often than general figurative expressions, which occurred in between 13.2% and 38.62% of sentences. In Figure 2b, we present the frequency of SE-specific figurative expressions identified within our corpus of 200k GitHub comments. Of the 445 SE-specific figurative expressions we investigated, 324 (72.8%) appear no more than 10 times, as indicated by the red dotted line, suggesting that most such expressions are infrequent. Among these, 193 expressions are absent from our dataset. This absence can be attributed to two main factors: a) expressions that are specific to particular projects (e.g., *'ghost highlight'* and *'ghost monitor'* in UI-related projects); and b) unique expressions used to describe highly specific scenarios (e.g., *'dead fork,' 'magic code'*).

Note that our study provides only a lower bound, as it matches using an incomplete set of figurative language expressions. Therefore, the likely presence of figurative language is even higher than we report. This exploratory study highlights the importance of understanding figurative language in the SE context, as it can provide insight into the daily communications of developers.



(a) Percent of sentences with figurative expressions in repos.



(b) Frequency of sentences with SE-specific figurative expressions.

Figure 2: Distribution of figurative language occurrence in GitHub sentences (200k GitHub comments, 484k sentences).

4 EXPERIMENTS AND DISCUSSION

Using the assembled dataset, we created specific experiments for each of our three research questions. In this section, we describe the experiments and discuss the corresponding results.

4.1 RQ1: How well can existing LLMs interpret figurative language (i.e., metaphors and idioms) used in software engineering?

In order to determine how well popular Large Language Models (LLMs) understand metaphors and idioms, we examine whether they can understand the semantic relationship between the original sentence, the equivalent sentence (i.e., EMS), and the different-meaning sentence (i.e., DMS). The task of differentiating EMSs from DMSs of the original sentences can be thought of as Recognizing Textual Entailment (RTE) [50]. RTE involves determining whether a statement, called the hypothesis, can be inferred from a given text, called the premise. In our context, the premise is the original sentence, and the hypotheses are the EMS and DMS. We evaluate whether an LLM can infer the EMS from the original sentence, and if it is a DMS, the model should not deduce it.

One way to approximate the RTE task is to posit that the model should recognize the original sentence as semantically closer to the EMS than to the DMS. By comparing the embedding vectors of the original sentence and the sentence in question, we can measure whether the two sentences are similar or dissimilar and, therefore, whether the most similar sentence is an EMS or a DMS.

4.1.1 Compared LLMs. We compare three LLMs – BERT [15], RoBERTa [16], and ALBERT [51], which are popular in NLP and SE tasks [39, 52, 53]. BERT [15] is a transformer-based model pre-trained on extensive text data from Wikipedia and BooksCorpus. RoBERTa [16] is an improved version of BERT, while ALBERT [51] enhances efficiency through parameter reduction techniques. Additionally, we evaluate a popular SE-specific LLM – CodeBERT [54], which is pre-trained on natural language - programming language pairs. We use *bert-base-uncased*, *roberta-base*, *albert-base-v2* and *microsoft/codebert-base* available from Hugging Face[20].

4.1.2 Procedure. We generate embedding vectors using each of the LLMs for each pair of sentences, i.e., <*original sentence, EMS*> and <*original sentence, DMS*>. Next, we compute the similarity between the vectors in each pair and then compare the resulting similarity scores [55]. We perform standard software engineering text-specific

[19]https://www.nltk.org/

[20]https://huggingface.co

Table 1: Percent of EMS with a higher similarity to the original sentence than corresponding DMS (Sim_{EMS} > Sim_{DMS}).

Model	*SE-specific*			*General*			*Overall*		
	Sim_{EMS} > Sim_{DMS}	*p-value*	|Cliff's δ|	Sim_{EMS} > Sim_{DMS}	*p-value*	|Cliff's δ|	Sim_{EMS} > Sim_{DMS}	*p-value*	|Cliff's δ|
BERT	84.51%	$p < 0.01$	0.629	87.40%	$p < 0.01$	0.638	86.57%	$p < 0.01$	0.637
RoBERTa	83.70%	$p < 0.01$	0.648	85.21%	$p < 0.01$	0.620	84.95%	$p < 0.01$	0.632
ALBERT	81.79%	$p < 0.01$	0.610	85.80%	$p < 0.01$	0.598	85.00%	$p < 0.01$	0.605
CodeBERT	77.99%	$p < 0.01$	0.498	79.63%	$p < 0.01$	0.493	79.11%	$p < 0.01$	0.495

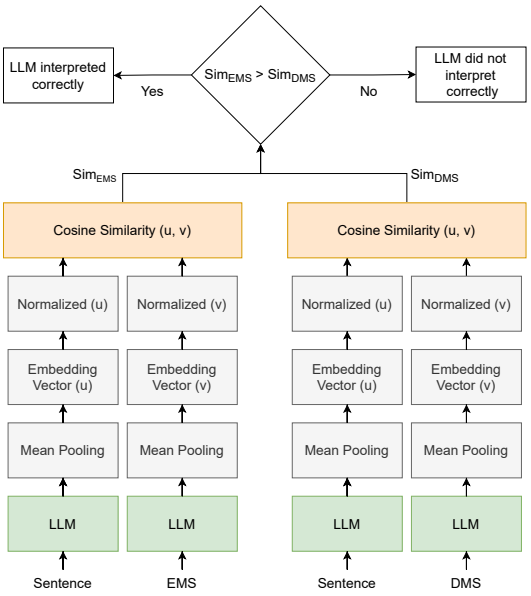


Figure 3: RQ1 evaluation pipeline.

preprocessing operations such as URL removal, username removal, stack trace removal, etc. [56]

Since the LLMs in our cohort produce word-level embedding vectors, there are several possibilities for aggregating these into sentence-level embedding vectors. Reimers et al. [55] noted that mean pooling (the mean of all per-word output vectors generated by the LLM) is one of the best strategies. While we opt for mean pooling when generating each sentence's embedding vector, we also note that this strategy is still error-prone due to the anisotropy problem, i.e., the difference in the scale of the embedding vectors [57]. For this reason, we apply the normalization proposed by Yan et al. [58], which is based on Singular Value Transformation (SVT). SVT uses singular value decomposition and a threshold using the soft-exponential function by Godfrey et al. [59]

Following normalization, we compute vector pair similarity with the cosine similarity metric. Cosine similarity measures vector alignment, with values from 1 (identical) to 0 (orthogonal) to -1 (opposite) [60]. Then, we compare the two similarities in order to determine if the *<original sentence, EMS>* similarity (Sim_{EMS}) is higher than the *<original sentence, DMS>* similarity (Sim_{DMS}). Figure 3 summarises the entire procedure. To evaluate the RQ, we compute the percentage of instances where Sim_{EMS} is greater than Sim_{DMS}. We examine three sentence categories: a) those containing SE-specific figurative language only (n=371); b) those containing general figurative language only (n=1179); and c) overall, containing either (a) and (b), or both (n=1661). For each model and category, we measure the statistical significance of the difference between the two cosine similarities using the one-tailed Wilcoxon signed-rank test (i.e., testing if Sim_{EMS} > Sim_{DMS} with statistical significance). We apply the Benjamini-Hochberg correction to control the false discovery rate. A small *p-value* (e.g., *p-value* < 0.05) indicates that the difference is unlikely to be due to chance and that there is a statistically significant difference between the two samples. We also compute the effect size, which measures the magnitude of the difference between the two samples, using Cliff's Delta (δ) [61], where $|\delta| > 0.147$, 0.33, and 0.474 indicate small, medium, and large effects respectively.

4.1.3 Results and Discussion. Table 1 shows the SE-specific, General, and Overall (i.e., combined) results. The higher the percentage of <*original sentence*, *EMS*> pairs with larger cosine similarity, i.e., Sim_{EMS} > Sim_{DMS}, the better the model is at recognizing figurative language. The results table shows that the BERT and RoBERTa models have the highest percentage of correctly understood pairs for all categories. BERT, RoBERTa, and ALBERT models correctly recognize 84.51%, 83.70%, and 81.79% of sentences containing SE-specific figurative expressions, 87.40%, 85.21%, and 85.80% of General figurative expressions, and 86.57%, 84.95%, and 85.0% of the Overall figurative expressions, respectively. In the case of CodeBERT, which exhibits the poorest results out of all models, there is no significant difference between SE-specific and General results (77.99% and 79.63% respectively). This is likely because the model is pre-trained with programming-specific data enabling it recognize some software engineering figurative language terms. However, it also likely loses the ability to capture General figurative language, which is present in the other LLMs. From this study, we observe that all of the models can understand figurative language to a reasonable degree (i.e., ranging between 77.99% to 87.40%).

This is also evident from *p-value* and Cliff's $|\delta|$. In each case, the statistically significant is with a *p-value* < 0.01 and a $|\delta|$ greater than 0.474 is considered a large effect for all models. The p-value less than 0.01 indicates that the observed difference in similarity between the two groups is highly unlikely to be due to chance and we conclude that there is a statistically significant difference in similarity between the two sets of sentence pairs. The large $|\delta|$ indicates that the similarity between the two groups (i.e., the sentence pairs in group Sim_{EMS} compared to those in group Sim_{DMS} is substantial. The cosine similarity values in Group Sim_{EMS} are consistently higher than those in Group Sim_{DMS}, showing that the sentences in Group Sim_{EMS} are more similar to each other compared to those in Group Sim_{DMS}. Together, these results suggest that the two groups of sentence pairs exhibit a notable and

meaningful difference in their similarity scores, and this difference is not likely to be due to random chance.

However, the models still fail to recognize between 18.21% and 12.60% of figurative language instances. It is highly likely that if we can improve the models' understanding of figurative language in such cases, they will function better in their use cases.

4.2 RQ2: Can the performance of software engineering-specific affective analysis be improved by a better insight into figurative language?

Affect analysis involves identifying and evaluating human emotions, feelings, and sentiments expressed through written communication. Kovecses et al. noted that figurative expressions are vital in expressing emotions [4], while Mohammad et al. [62] observed that metaphorical words tend to contain significantly more emotions than the literal sense of the same words. In software engineering, affect is often related to the software and its development process, including the emotional states of software developers, productivity, and burnout [63–65]. Thus, identifying and understanding affect is crucial for improving software quality and developer productivity. However, several studies have shown challenges in building reliable tools and datasets for mining emotions and opinions in the SE domain [33, 66]. A recent study by Imran et al. found that using figurative language in SE-related text can hinder the accurate identification of emotions [10], partly motivating our RQ2 investigation.

The use of LLMs has become a widely adopted method for identifying and classifying affective expressions in written text [67–69]. LLMs are usually fine-tuned to address specific affect analysis tasks, such as recognizing sentiment or emotions. Recently, one of the most effective ways to fine-tune an LLM is by applying a contrastive learning approach. This approach uses sets of similar and dissimilar instances to train the model to understand the similar instances and differentiate them from the dissimilar ones [70]. To answer this RQ, we leverage contrastive learning as the means for LLMs to better capture the meaning and nuances in figurative language present in GitHub comments.

4.2.1 Compared models. Similar to RQ1, we assess the ability of the same four LLMs — BERT [15], RoBERTa [16], ALBERT [51], CodeBERT [54] — with the same model versions as RQ1 from Hugging Face. Previous research shows that BERT, RoBERTa and ALBERT work well in SE affect analysis [21, 34, 52].

4.2.2 Contrastive learning. Contrastive learning is a recently proposed machine learning technique that involves training a model to distinguish between two or more distinct data points by contrasting their differences [31, 71]. The steps for applying this approach to fine-tune LLMs for understanding figurative language elements in the text can be outlined as follows:

(1) The LLM is presented with a triplet of anchor, positive and negative samples, which are representative of the figurative language elements to be learned.
(2) The LLM processes the samples and generates output embeddings for the data triplet.
(3) A loss function encourages the anchor and positive samples to be closer together and the anchor and negative samples to be further apart in the embedding space.
(4) The process is repeated until the LLM has learned a satisfactory representation.

To apply contrastive learning, we use the original sentences and EMSs as anchor and positive classes and DMSs as negative classes. In other words, we created *<original sentence, EMS, DMS>* and *<EMS, original sentence, DMS>* as a pair of triplets, where the first element in each pair is anchor, the second element is positive, and the third element is negative. There is a total of 3322 such triplets of sentences in our dataset.

We use InfoNCE Loss as our loss function [31]. Given the embeddings of an anchor, a positive, and a negative sample denoted as a, p, and n respectively, the InfoNCE loss is computed as follows: $\text{InfoNCE Loss}(a, p, n) = -\log\left(\frac{e^{\text{sim}(a,p)}}{e^{\text{sim}(a,p)}+e^{\text{sim}(a,n)}}\right)$ where $\text{sim}(a, p)$ represents the cosine similarity between the embeddings of the anchor and positive samples, and $\text{sim}(a, n)$ represents the cosine similarity between the embeddings of the anchor and negative samples. The InfoNCE loss maximizes the log-likelihood of anchor-positive similarity and minimizes anchor-negative similarity. We use the Adam optimizer.

Using contrastive learning, the LLM learns to create embeddings that capture the semantic similarity between the original and EMS while recognizing the semantic differences between the original and DMS. This allows the LLM to learn a representation that separates the positive and negative samples as much as possible. In this case, the LLM learns to recognize the figurative language elements.

After fine-tuning the models with contrastive learning, we assess their performance in two tasks: emotion recognition and incivility detection. We compare the performance of these fine-tuned models against baseline models that are not fine-tuned with figurative language.

4.2.3 Datasets. We apply the LLMs to two SE-affect datasets.

Emotion Dataset. Imran et al. [10] curated a multi-label emotion dataset that is crawled from GitHub. The dataset consists of 2000 data points and six emotion classes: Anger, Love, Fear, Joy, Sadness, and Surprise. The dataset contains 340 (17.0%) Anger comments, 220 (11.0%) Love comments, 198 (9.9%) Fear comments, 422 (21.1%) Joy comments, 274 (13.7%) Sadness comments, and 328 (16.4%) Surprise comments. The rest of the comments are neutral.

Incivility Dataset. Ferreira et al. [72] curated a dataset from GitHub's heated issues for incivility detection. The dataset has three parts: comment level, issue level, and sentence level. We consider only the comment-level dataset in our study, which has three classes: Civil, Uncivil, and Technical. We consider only Civil and Uncivil comments as we are only interested in affective analysis for this RQ. The filtered dataset contains 718 comments, of which 232 (32.3%) are Civil comments, and 486 (67.7%) Uncivil comments.

4.2.4 Procedure and Metrics. Using random stratified sampling for each class, we divide all two datasets into train (80%) and test (20%) sets [73]. For each task (i.e., incivility detection and emotion detection), we train (or fine-tune) both the LLMs' contrastive learning and baseline versions. In other words, the contrastive learning models are fine-tuned twice, first with contrastive learning and

figurative language and second with a task-specific dataset. The baselines are only fine-tuned with the task-specific dataset.

We choose a metric that is frequently used to evaluate classification tasks: *F1-score*, which aggregates *Precision* and *Recall*. Precision is the ratio of true positive instances to the total predicted positive instances, and Recall is the ratio of true positive instances to all instances in the positive class. *F1-score* is the harmonic mean of *Precision* and *Recall*: $F1\text{-}score = 2 * \frac{Precision * Recall}{Precision + Recall}$. We also calculate the micro average version for averaging the F1-score across the classes following previous research [10, 74].

4.2.5 Results and Discussion. ***Emotion Classification:*** Table 2 shows the results of the emotion classification task on Imran et al.'s multi-label emotion dataset [10], using BERT, RoBERTa (RBTa), ALBERT (ALBT), CodeBERT (CodBT) and their fine-tuned with figurative language counterparts (BERT-FL, RBTa-FL, ALBT-FL, CodBT-FL). The table presents the F1-score for each emotion class, the micro-averaged F1-score, and the improvement in the F1-score achieved by the figurative language versions of the models. The results show that the use of contrastive learning with figurative language improves performance on the emotion classification task for most emotion types. For the micro-averaged F1-score, across all emotions, the figurative language versions of the models achieve an improvement of 6.60%, 6.66%, 3.63%, and 3.90% for BERT, RoBERTa, ALBERT, and CodeBERT respectively. This implies that adding figurative language to these models improves their capability to comprehend and interpret the subtleties that developers use in their communication.

In all four models, we see an increase in True Positives and a decrease in both False Negatives and False Positives. For instance, for the BERT model, across 6 emotions, micro-averaged recall increases by 5.58%, and micro-averaged precision increases by 7.59%. This indicates improved precision in predictions after applying contrastive learning.

When considering the average improvement in individual emotions across all models in Table 2, we observe that *'Joy'* has most improvement (7.67%), followed by *'Surprise'* (5.45%). This correlates with the frequency of occurrence of these emotions in GitHub comments, e.g., Joy is much more commonly found than Fear. As our figurative language dataset is randomly sampled, it is likely to contain figurative expressions closely related to the emotions that are more commonly observed in GitHub. This result suggests that curating a larger and more diverse set of comments that include figurative language could lead to a stronger and more balanced performance improvement.

Error Analysis of BERT-FL vs. BERT. To gain deeper insight into figurative language-based models' predictive accuracy relative to baseline models, we perform qualitative analysis. Our focus is solely on BERT and BERT-FL models' predictions. We examine two specific areas: 1) True Positives where BERT-FL is correct while baseline BERT is not, and 2) True Positives where baseline BERT is correct while BERT-FL is not.

Among the positive instances, BERT-FL correctly predicts 39 utterances that the baseline BERT model does not. Consider the following sentence: *"Bah. Wasn't supposed to add anything – it was a debugging leftover..."*. In this case, BERT-FL correctly predicts *'Anger'*, whereas BERT misclassifies it. Here, the word *'leftover'* is used metaphorically. Normally, *'leftover'* refers to *'something that remains unused or unconsumed'*, particularly in the context of food[21]. However, in the given sentence, the word is used to imply that some code or modifications were unintentionally left behind or overlooked during the debugging process. The BERT-FL model likely captures the context more effectively. Another example, *"Oh nice!! I've seen that syntax floating around, wanting to try it for a while* 🙌*"* - BERT-FL correctly classifies as *'Joy'* which the BERT baseline model misclassifies. Here, the BERT-FL is likely able to capture that *'floating around'* is an idiom[22] and interpret the meaning. In some instances, BERT-FL makes correct predictions by adopting a more conservative classification approach. For instance, BERT classifies the following sentence as *'Anger'*: *"Please put this below line 5 (together with the other non-app imports) :pray:"*. However, BERT-FL accurately predicts that it is not *'Anger'*.

Table 2: Evaluation of LLMs finetuned with figurative language on the Emotions Dataset (F1-score).

Model	Anger	Love	Fear	Joy	Sad.	Surp.	Mic.Avg.
BERT	0.506	**0.712**	0.536	0.579	0.636	0.594	0.588
BERT-FL	**0.547**	0.709	**0.562**	**0.608**	**0.661**	**0.632**	**0.627**
+/-	+8.10%	-0.42%	+4.85%	+5.01%	+3.93%	+6.40%	+6.60%
RoBERTa	0.525	0.683	0.500	0.613	**0.673**	0.592	0.593
RoBERTa-FL	**0.551**	**0.733**	**0.545**	**0.667**	0.667	**0.617**	**0.632**
+/-	+4.95%	+6.82%	+8.26%	+8.10%	-0.90%	+4.05%	+6.66%
ALBERT	**0.462**	0.658	0.430	0.487	**0.628**	0.564	0.531
ALBERT-FL	0.443	**0.682**	**0.435**	**0.540**	0.624	**0.592**	**0.550**
+/-	-4.11%	+3.52%	+1.15%	+9.81%	-0.64%	+4.73%	+3.63%
CodeBERT	0.484	0.711	**0.507**	0.558	0.575	0.576	0.561
CodeBERT-FL	**0.497**	**0.723**	0.444	**0.605**	**0.645**	**0.617**	**0.583**
+/-	+2.79%	+1.70%	-14.08%	+7.75%	+10.92%	+6.61%	+3.90%
Avg. +/-	+2.93%	+2.90%	+0.04%	+7.67%	+3.33%	+5.45%	+5.20%

On the other hand, in 27 cases, BERT-FL makes wrong predictions where BERT does not. Consider this utterance: *"I have currently no clue, but I'll have a look"*, this sentence contains the idioms *'have a clue'* [23] and *'have a look'* [24]. The author of the comment likely was puzzled about some functions or errors. BERT identifies correctly as *'Surprise'* but BERT-FL does not. Possibly, BERT-FL interpreted these idiomatic expressions more of a literal interpretation of the words. In some cases, BERT-FL just misclassifies without any involvement of any figurative expressions. For example, *"I guess my concern is that it sets a precedent where somebody could see it and think that it would be fine to use in 'core'."* This expression express concern which is annotated as *'Fear'*. This expression conveys concern, annotated as *'Fear'*. BERT identifies it correctly, but BERT-FL does not. It is possible that during the contrastive learning process, BERT-FL may lose some of the baseline BERT model's ability to capture nuanced emotional indicators in certain sentences accurately. This suggests that while this approach improves the overall model performance but may introduce limitations or biases in some cases.

Incivility Classification: Table 3 presents the results of the *incivility classification* task on Ferreira et al.'s incivility dataset [72], using the same four large language models (BERT, RoBERTa, ALBERT, and CodeBERT) with and without the contrastive learning approach. The micro-averaged F1-scores indicate that the models perform better when applying the contrastive learning approach.

[21] https://www.merriam-webster.com/dictionary/leftover
[22] https://idioms.thefreedictionary.com/floating+around
[23] https://idioms.thefreedictionary.com/have+a+clue
[24] https://www.thefreedictionary.com/have+a+look

Table 3: Evaluation of LLMs finetuned with figurative language on the Incivility Dataset (F1-score).

Model	Civil	Uncivil	Micro Average
BERT	0.537	0.814	0.734
BERT-FL	**0.587**	**0.853**	**0.783**
+/-	+8.54%	+4.84%	+6.67%
RoBERTa	0.424	0.827	0.734
RoBERTa-FL	**0.535**	**0.847**	**0.769**
+/-	+20.73%	+2.33%	+4.76%
ALBERT	0.151	0.807	0.685
ALBERT-FL	**0.423**	**0.809**	**0.713**
+/-	+64.28%	+0.30%	+4.08%
CodeBERT	0.185	0.810	0.692
CodeBERT-FL	**0.431**	**0.833**	**0.741**
+/-	+57.01%	+2.74%	+7.07%
Avg. +/-	+37.64%	+2.55%	+5.65%

Table 4: Evaluation of LLMs finetuned with figurative language on the Bug Report Priority dataset (F1-score).

Model	P1	P2	P3	P4	P5	Micro Average
BERT	0.606	0.329	0.833	0.0	0.663	0.716
BERT-FL	**0.632**	**0.359**	**0.842**	0.0	**0.667**	**0.730**
+/-	+4.31%	+9.14%	+1.10%	-	+0.52%	+1.96%
RoBERTa	0.61	0.293	0.827	0.0	**0.677**	0.707
RoBERTa-FL	**0.624**	**0.343**	**0.839**	0.0	0.674	**0.724**
+/-	+1.91%	17.24%	+1.39%	-	-0.51%	+2.40%
ALBERT	0.564	0.288	0.810	0.0	0.670	0.683
ALBERT-FL	**0.602**	**0.299**	**0.827**	0.0	**0.674**	**0.709**
+/-	+6.71%	+3.88%	+2.14%	-	+0.53%	+3.71%
CodeBERT	0.608	0.363	0.830	0.0	0.667	0.714
CodeBERT-FL	**0.636**	**0.373**	**0.839**	0.0	**0.670**	**0.726**
+/-	+4.55%	2.64%	+1.08%	-	0.52%	+1.61%
Avg. +/-	+4.37%	+8.23%	+1.43%	-	+0.27%	+2.42%

Overall, the BERT, RoBERTa, ALBERT, and CodeBERT models have an average improvement of 6.67%, 4.76%, 4.08%, and 7.07% respectively, when the contrastive learning approach is applied. Since the incivility dataset is small and imbalanced, the baseline models often struggle to classify the minor *'Civil'* class, except for BERT.

We also observed a significant average improvement of 37.64% across all models in the *'Civil'* class, compared to a modest 2.55% improvement in the *'Uncivil'* class. This discrepancy likely arises because the figurative language dataset used for contrastive learning primarily consists of *'Civil'* comments, which are much more common on GitHub than *'Uncivil'* comments. Incorporating more figurative expressions from *'Uncivil'* comments into the dataset could potentially enhance performance in this category as well.

It is important to note that the substantial improvements in identifying *'Civil'* comments are largely attributable to ALBERT and CodeBERT, which showed improvements of 180% and 133%, respectively. These models started from a lower performance baseline, making such large gains more achievable compared to other models. However, BERT and RoBERTa also demonstrated stronger performance improvements in the *'Civil'* class.

4.3 RQ3: Can a better understanding of figurative language enhance software engineering automation where affect plays a role?

To answer this RQ, we focus on a specific use case: automatic bug report priority detection, a major research area in software engineering [34, 35, 37, 75], where previous research has highlighted the role of affect [36].

Dataset. Bugzilla bug reports are widely used for priority detection [34, 36, 75]. The bug priority reports in Bugzilla are divided into 5 classes (i.e., P1 to P5, where P1 represents the highest priority while P5 represents the lowest priority). Wang et al. collected 220k bug reports from Bugzilla [34]. We sample 25% of this dataset using stratified sampling across the 5 classes. We sample separately from the training and testing splits provided by the authors, which yielded a total of 49.6k bug reports. The distributions provided by the authors are: 1) training: P1 - 19.56%, P2 - 18.45%, P3 - 58.12%, P4 - 1.66%, and P5 - 2.21%; and 2) testing: P1 - 19.21%, P2 - 17.66%, P3 - 59.5%, P4 - 1.48%, and P5 - 2.15%.

Procedure and Metrics. We use the same four LLMs (BERT, RoBERTa, ALBERT, and CodeBERT) as baselines and follow the same approach for training and testing described in RQ2. We use F1-score as evaluation metric.

Results and Discussion. Table 4 shows the results of bug report priority prediction on the Bugzilla dataset. All four models made small improvements (1.96%, 2.40%, 3.71%, and 1.61% respectively) when fine-tuned with figurative languages. On the other hand, the improvements across classes (P1-P5) varied. The change in the P5 class was minimal (0.27%), and none of the models succeeded in recognizing any of the P4 instances. This is likely due to the fact that these two classes have the smallest amounts of data, comprising only 1.66% for P4 and 2.21% for P5 of the training data, respectively. Such findings suggest that fine-tuning with figurative language is not beneficial in cases of extreme data imbalance. For the average performance improvement across all models in the other three bug priority classes, we observe that P3 improved least (1.43%) while P1 and P2 make more substantial gains of 4.37% and 8.23%. Umer et al. [36] noted that a substantial number of instances in the Bugzilla dataset are *'Neutral'*, indicating that including figurative expressions from *'Neutral'* utterances — which our dataset predominantly omits — could potentially yield additional benefits.

Error Analysis of BERT-FL vs. BERT. To get an understanding of where fine-tuned models are getting results correctly compared to baseline models, we look into 51 instances where BERT-FL makes the right predictions but BERT does not. We find that, indeed, some of these bug reports include metaphors and idioms. For example, consider the following bug report description, which is at the P2 priority level: *"Deadlock when adding JSF framework I have experienced a deadlock while I was adding JSF framework to regular web project. [...]"* Here *'Deadlock'* is a SE-specific figurative expression. The baseline model predicted P3, but BERT-FL made the correct prediction. Another example *"Toot your own horn, put your name in the credits window The credits window is empty [...]"*, annotated as P3. Here, *'toot your own horn'* is an idiom. BERT-FL correctly predicted but the baseline model did not.

However, there are also cases with figurative language where the fine-tuned model predicted incorrectly, while the baseline model was right. For example, consider the following bug report *"offline task data is not retrieved on query [...] (i.e., fetch all things before hitting the road). [...]"* Here, *'hitting the road'* is an idiom[25]. The BERT-FL model predicts P1 when the original label is P3. It is possible that BERT-FL recognizes the idiom, prioritizes its figurative meaning, and predicts a higher class than the original label.

[25]https://www.thefreedictionary.com/hitting+the+road

4.4 Implications

There are a number of actionable implications to our study. Creating a glossary of common figurative language for a software project can be an invaluable tool for efficiently onboarding new developers [76]. It would help newcomers understand project-specific or domain-specific terms, which are essential for their quick integration. Minimizing the use of obscure jargon that may cause misunderstandings can enhance mutual understanding and collaboration among project participants [77]. Lastly, it's important to consider cultural differences [77] that may influence the interpretation of figurative language, as these nuances can significantly affect comprehension and communication within a diverse team.

Our study paves the way for several promising research directions in the realm of figurative language comprehension within software engineering: 1) Integrating figurative language into cutting-edge software engineering tools, such as CleBPI [34], could be achieved through innovative approaches like contrastive learning, self-supervised learning, or adversarial training; 2) Investigating the role of figurative language in specific scenarios, including toxic or uncivil comments, bug reports, and documentation, may yield insights into its effects on software development workflows; 3) Exploring the use of figurative language as a means for data augmentation presents an intriguing opportunity, building on established data augmentation techniques [10]; 4) Broadening the scope of analysis to encompass various forms of figurative language, such as similes, hyperbole, and personification, could enhance the depth of model training; 5) Extending our analysis to software engineering communication platforms beyond GitHub, including Stack Overflow, Gitter, JIRA, and app reviews, would offer a more holistic view of figurative language usage across different settings. Adapting Large Language Models (LLMs) for domain-specific figurative language has recently garnered interest in the NLP community [78–81]. Our work compliments this by adapting LLMs to the figurative language in software engineering.

5 RELATED WORK

We describe the related work sourced from three different domains: figurative language analysis in the domain of Natural Language Processing (NLP), affect analysis in Software Engineering (SE), and bug report analysis in SE.

Figurative Language in NLP. Figurative language has long been a topic of study in the field of NLP [27, 50, 82, 83]. Research has explored its impact across various communication channels, including online reviews and social media [84, 85]. Social media platforms frequently employ figurative language to convey emotions, opinions, and feelings [86]. Furthermore, there have been investigations that focus on the use of figurative language within specific domains [78–80]. Specific forms of figurative language, such as metaphors, idioms, similes, sarcasm, and irony, have been studied in relation to tasks like offensive language detection [5, 87].

Scholars have categorized the detection of figurative language into two tasks: recognizing text containing figurative language and interpreting figurative expressions to identify their intended literal meaning [88]. Recognizing figurative language presents challenges due to the multiple interpretations that expressions can have. To address these challenges, various methods have been proposed, such as word vectors, rule-based approaches, semantic patterns, and the application of LLMs [23, 24, 89, 90]. Interpreting figurative language is a more complex task that requires a deeper understanding of the text's meaning [91]. Previous approaches have utilized knowledge-based and corpus-based methods [92, 93]. Researchers have leveraged LLMs to paraphrase figurative expressions tasks and have been successful in interpreting metaphors, idioms, hyperbole, irony, sarcasm, and similes [27, 41, 50, 78]. Some of the most common strategies applied to interpret figurative expressions using LLMs are zero-shot learning and fine-tuning using contrastive learning [70]. Different from the prior work, we investigate the ability of LLMs to interpret figurative language in the context of SE communication.

Affect Analysis in SE. Affective expressions in written text can be effectively analyzed by identifying linguistic cues that convey emotions, feelings, or attitudes [94, 95]. The field of affect analysis in software engineering is rapidly growing, focusing on understanding how emotions, opinions, sentiment, toxicity, incivility, burnout, and offensive language impact software development activities [6, 10, 11, 19, 22, 32, 33, 38, 66, 72, 96–101].

LLMs have emerged as powerful tools for affect analysis, making significant strides in the software engineering domain [67–69, 102]. They have proven their mettle in sentiment analysis across various software-related artifacts and have even been instrumental in detecting incivility and toxicity [21, 39, 101, 103].

Despite the advancements made and the use of modern LLMs, some limitations need to be addressed, particularly in terms of generalizability. SE-specific affect analysis tools trained on one communication forum may not perform well when applied to another forum due to differences in norms, conventions, and cultures that influence the expression of emotions and sentiments [8, 9, 66]. One major reason for this limitation is the tools' inability to recognize implicit emotions or sentiments, often inferred through context, tone, or other cues [10, 66]. These challenges call for developing more versatile and adaptable tools that can be applied across multiple domains. In this paper, we explore interpreting and fine-tuning figurative languages with LLMs to enhance the generalizability of SE-specific affect analysis tools across different artifacts.

Bug Report Analysis in SE. Bug report analysis is a mature research area in SE spanning tasks like duplicate bug detection, bug localization, deficient bug report, bug severity prediction, and priority assignment [17, 34, 37, 56, 75, 104–106]. Of particular relevance is bug report priority prediction, where affects in report descriptions can influence triage decisions [36]. Recently, priority inference models based on deep learning have been proposed using LLMs [34]. This study explores whether fine-tuning LLMs with figurative language can enhance performance of the task or not.

6 THREATS TO VALIDITY

Several limitations may impact the interpretation of our findings. We categorize and list each of them below.

Construct validity. Construct validity refers to the degree to which the study measures the concepts and constructs it claims to measure. A threat may arise from the manual annotations for the dataset, specifically in creating semantically similar EMS and

DMS sentences. To mitigate this, we provided clear instructions and examples to the annotators. Additionally, we only examined metaphors and idioms; including other figurative language may alter results. To investigate this, our annotation approach can be expanded to analyze other forms. Another potential threat is that our figurative language dataset was sourced from developer communication in 9 GitHub repositories, which may not be representative of the figurative language present on GitHub.

Internal validity. Internal validity concerns the extent to which the study's findings can be attributed to the manipulation of the independent variable. A threat is that the improved affect analysis performance with figurative language fine-tuning may not be solely due to the figurative language. However, we see consistent improvements across all models and datasets, indicating it is a key factor. Not doing cross-validation on the smaller datasets can be another threat. To mitigate this, we use stratified sampling for representativeness and a standard 80-20% train-test split.

External validity. External validity pertains to the generalization of the findings of our study to other settings and contexts. Our results may not generalize beyond the specific studied models, datasets, and any other domain than GitHub. However, we use diverse pre-trained LLMs and a Bugzilla dataset, showing some cross-domain applicability. Further investigation is needed to validate our results beyond the tools, data, and platforms used in our study.

7 CONCLUSION

This paper examined the relevance and impact of figurative language in software engineering communication. To investigate this, we annotated metaphors and idioms in a set of 2000 sentences collected from GitHub issues and PRs which resulted in 1661 sentences with figurative expressions, conducted a comprehensive analysis of the prevalence of figurative language in messages posted on PRs and issues in top 100 GitHub repositories, fine-tuned several state-of-the-art pre-trained LLMs with the annotated dataset, and evaluated the performance of these fine-tuned models on three publicly available SE-specific datasets. Our results indicated that figurative language is prevalent in software engineering communication, and fine-tuning LLMs with figurative language leads to improved performance on affect analysis tasks (on the best model, 6.66% improvement on a GitHub emotion dataset, 7.07% improvement on a GitHub incivility dataset, and 3.71% improvement on a bug report prioritization dataset). Overall, our findings provide evidence for the relevance and impact of figurative language in software engineering communication and the potential benefits of fine-tuning LLMs with figurative language in the context of software engineering. However, there is room for further investigation.

Beyond the future work directions discussed in Section 4.4, our error analysis shows that fine-tuned models may sometimes overemphasize figurative language, motivating the need for a different fine-tuning approach. Addressing this issue while preserving interpretive abilities presents an area for future research. Experimenting with generative language models like ChatGPT and LLaMa to assess their potential in enhancing the automatic interpretation of complex figurative expressions could significantly benefit communication and understanding in software development contexts. Overall, this study provides a starting point for further empirical research on figurative language's impact on software engineering communications in different application domains.

REFERENCES

[1] R. Giora and O. Fein, "On understanding familiar and less-familiar figurative language," *Journal of Pragmatics*, vol. 31, no. 12, pp. 1601–1618, 1999.
[2] L. Zhang, Y. Sun, H. Song, W. Wang, and G. Huang, "Detecting anti-patterns in java ee runtime system model," in *Proceedings of the Fourth Asia-Pacific Symposium on Internetware*. USA: ACM, 2012.
[3] G. L. Steele, "Macaroni is better than spaghetti," in *Proceedings of the 1977 Symposium on Artificial Intelligence and Programming Languages*. USA: ACM, 1977, p. 60–66.
[4] Z. Kövecses, "Emotion concepts: Social constructionism and cognitive linguistics," in *The verbal communication of emotions*. Psychology Press, 2002, pp. 117–132.
[5] F. M. P. del Arco, M. D. Molina-González, L. A. Ureña-López, and M.-T. Martín-Valdivia, "Integrating implicit and explicit linguistic phenomena via multi-task learning for offensive language detection," *Knowledge-Based Systems*, vol. 258, p. 109965, 2022.
[6] I. Ferreira, J. Cheng, and B. Adams, "The "shut the f**k up" phenomenon: Characterizing incivility in open source code review discussions," *Proc. ACM Hum.-Comput. Interact.*, vol. 5, no. CSCW2, oct 2021.
[7] D. Tiwari, T. Toady, M. Monperrus, and B. Baudry, "With great humor comes great developer engagement," *ArXiv*, vol. abs/2312.01680, 2023.
[8] N. Novielli, F. Calefato, F. Lanubile, and A. Serebrenik, "Assessment of off-the-shelf se-specific sentiment analysis tools: An extended replication study," *Empirical Softw. Engg.*, vol. 26, no. 4, jul 2021.
[9] 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 *2020 IEEE/ACM 17th International Conference on MSR*. USA: IEEE Computer Society, may 2020, pp. 158–168.
[10] M. M. Imran, Y. Jain, P. Chatterjee, and K. Damevski, "Data augmentation for improving emotion recognition in software engineering communication," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. USA: ACM, 2023.
[11] Z. Chen, Y. Cao, H. Yao, X. Lu, X. Peng, H. Mei, and X. Liu, "Emoji-powered sentiment and emotion detection from software developers' communication data," *ACM TOSEM*, vol. 30, no. 2, pp. 1–48, 2021.
[12] X. Chen, C. Chen, D. Zhang, and Z. Xing, "Sethesaurus: Wordnet in software engineering," *IEEE Transactions on Software Engineering*, vol. 47, no. 9, pp. 1960–1979, 2019.
[13] C. Chen, Z. Xing, and X. Wang, "Unsupervised software-specific morphological forms inference from informal discussions," in *2017 IEEE/ACM 39th ICSE*. USA: IEEE Computer Society, may 2017, pp. 450–461.
[14] C. V. Alexandru, J. J. Merchante, S. Panichella, S. Proksch, H. C. Gall, and G. Robles, "On the usage of pythonic idioms," in *Proceedings of the 2018 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*. USA: ACM, 2018, p. 1–11.
[15] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the NAACL: Human Language Technologies*. Minneapolis, Minnesota: ACL, Jun. 2019, pp. 4171–4186.
[16] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *ArXiv*, vol. abs/1907.11692, 2019.
[17] A. Ciborowska and K. Damevski, "Fast changeset-based bug localization with bert," in *Proceedings of the 44th ICSE*. USA: ACM, 2022, p. 946–957.
[18] M. Ciniselli, N. Cooper, L. Pascarella, D. Poshyvanyk, M. D. Penta, and G. Bavota, "An empirical study on the usage of BERT models for code completion," in *18th IEEE/ACM International Conference on MSR*. Madrid, Spain: IEEE, 2021, pp. 108–119.
[19] B. Lin, F. Zampetti, G. Bavota, M. Di Penta, M. Lanza, and R. Oliveto, "Sentiment analysis for software engineering: How far can we go?" in *Proceedings of the 40th ICSE*. USA: ACM, 2018, p. 94–104.
[20] O. Sghaier and H. Sahraoui, "A multi-step learning approach to assist code review," in *2023 IEEE International Conference on SANER*. USA: IEEE Computer Society, mar 2023, pp. 450–460.
[21] T. Zhang, B. Xu, F. Thung, S. Haryono, D. Lo, and L. Jiang, "Sentiment analysis for software engineering: How far can pre-trained transformer models go?" in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. USA: IEEE Computer Society, oct 2020, pp. 70–80.
[22] J. Sarker, A. Turzo, and A. Bosu, "A benchmark study of the contemporary toxicity detectors on software engineering interactions," in *2020 27th Asia-Pacific Software Engineering Conference (APSEC)*. USA: IEEE Computer Society, dec 2020, pp. 218–227.

[23] C. Su, F. Fukumoto, X. Huang, J. Li, R. Wang, and Z. Chen, "DeepMet: A reading comprehension paradigm for token-level metaphor detection," in *Proceedings of the Second Workshop on Figurative Language Processing*. Online: ACL, Jul. 2020, pp. 30–39.

[24] G. Gamage, D. D. Silva, A. Adikari, and D. Alahakoon, "A bert-based idiom detection model," in *15th International Conference on Human System Interaction*. Melbourne, Australia: IEEE, 2022, pp. 1–5.

[25] J. Briskilal and C. Subalalitha, "An ensemble model for classifying idioms and literal texts using bert and roberta," *Information Processing & Management*, vol. 59, no. 1, p. 102756, 2022.

[26] W. Song, S. Zhou, R. Fu, T. Liu, and L. Liu, "Verb metaphor detection via contextual relation learning," in *Proceedings of the 59th Annual Meeting of the ACL and the 11th IJCNLP*. Online: ACL, Aug. 2021, pp. 4240–4251.

[27] K. Stowe, P. Utama, and I. Gurevych, "IMPLI: Investigating NLI models' performance on figurative language," in *Proceedings of the 60th Annual Meeting of the ACL*. Dublin, Ireland: ACL, May 2022, pp. 5375–5388.

[28] S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning, "A large annotated corpus for learning natural language inference," in *Proceedings of the 2015 Conference on EMNLP*. Lisbon, Portugal: ACL, Sep. 2015, pp. 632–642.

[29] S. R. Bowman, J. Palomaki, L. Baldini Soares, and E. Pitler, "New protocols and negative results for textual entailment data collection," in *Proceedings of the 2020 Conference on EMNLP*. Online: ACL, Nov. 2020, pp. 8203–8214.

[30] S. M. Mohammad, "9 - sentiment analysis: Detecting valence, emotions, and other affectual states from text," in *Emotion Measurement*, H. L. Meiselman, Ed. United Kingdom: Woodhead Publishing, 2016, pp. 201–237.

[31] A. v. d. Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *ArXiv*, vol. abs/1807.03748, 2018.

[32] P. Chatterjee, K. Damevski, and L. L. Pollock, "Automatic extraction of opinion-based q&a from online developer chats," in *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021*. Madrid, Spain: IEEE, 2021, pp. 1260–1272.

[33] B. Lin, N. Cassee, A. Serebrenik, G. Bavota, N. Novielli, and M. Lanza, "Opinion mining for software development: A systematic literature review," *ACM Trans. Softw. Eng. Methodol.*, vol. 31, no. 3, mar 2022.

[34] W.-Y. Wang, C.-H. Wu, and J. He, "Clebpi: Contrastive learning for bug priority inference," *Information and Software Technology*, vol. 164, p. 107302, 2023.

[35] Y. Tian, D. Lo, X. Xia, and C. Sun, "Automated prediction of bug report priority using multi-factor analysis," *Empirical Software Engineering*, vol. 20, no. 5, pp. 1354–1383, 10 2015.

[36] Q. Umer, H. Liu, and Y. Sultan, "Emotion based automated priority prediction for bug reports," *IEEE Access*, vol. 6, pp. 35 743–35 752, 2018.

[37] Q. Umer, H. Liu, and I. Illahi, "Cnn-based automatic prioritization of bug reports," *IEEE Transactions on Reliability*, vol. 69, no. 4, pp. 1341–1354, 2019.

[38] A. Murgia, P. Tourani, B. Adams, and M. Ortu, "Do developers feel emotions? an exploratory analysis of emotions in software artifacts," in *Proceedings of the 11th Working Conference on MSR*. USA: ACM, 2014, p. 262–271.

[39] E. Biswas, M. Karabulut, L. Pollock, and K. Vijay-Shanker, "Achieving reliable sentiment analysis in the software engineering domain using bert," in *2020 IEEE ICSME*. USA: IEEE Computer Society, oct 2020, pp. 162–173.

[40] Y. Nie, A. Williams, E. Dinan, M. Bansal, J. Weston, and D. Kiela, "Adversarial NLI: A new benchmark for natural language understanding," in *Proceedings of the 58th Annual Meeting of the ACL*. Online: ACL, Jul. 2020, pp. 4885–4901.

[41] T. Chakrabarty, A. Saakyan, D. Ghosh, and S. Muresan, "FLUTE: Figurative language understanding through textual explanations," in *Proceedings of the 2022 Conference on EMNLP*. Abu Dhabi, United Arab Emirates: ACL, Dec. 2022, pp. 7139–7159.

[42] P. Group, "Mip: A method for identifying metaphorically used words in discourse," *Metaphor and Symbol*, vol. 22, no. 1, pp. 1–39, 2007.

[43] E.-L. Do Dinh, H. Wieland, and I. Gurevych, "Weeding out conventionalized metaphors: A corpus of novel metaphor annotations," in *Proceedings of the 2018 Conference on EMNLP*. Brussels, Belgium: ACL, Oct.-Nov. 2018, pp. 1412–1424.

[44] J. Zhou, H. Gong, and S. Bhat, "PIE: A parallel idiomatic expression corpus for idiomatic sentence generation and paraphrasing," in *Proceedings of the 17th Workshop on Multiword Expressions (MWE 2021)*. Online: ACL, Aug. 2021, pp. 33–48.

[45] H. Haagsma, J. Bos, and M. Nissim, "MAGPIE: A large corpus of potentially idiomatic expressions," in *Proceedings of the Twelfth Language Resources and Evaluation Conference*. Marseille, France: European Language Resources Association, May 2020, pp. 279–287.

[46] OpenAI, "Gpt-4 technical report," *ArXiv*, vol. abs/2303.08774, 2023.

[47] F. Gilardi, M. Alizadeh, and M. Kubli, "Chatgpt outperforms crowd workers for text-annotation tasks," *Proceedings of the National Academy of Sciences*, vol. 120, no. 30, p. e2305016120, 2023.

[48] F. Huang, H. Kwak, and J. An, "Is chatgpt better than human annotators? potential and limitations of chatgpt in explaining implicit hate speech," in *Companion Proceedings of the ACM Web Conference 2023*. USA: ACM, 2023, p. 294–297.

[49] J. Kocoń, I. Cichecki, O. Kaszyca, M. Kochanek, D. Szydło, J. Baran, J. Bielaniewicz, M. Gruza, A. Janz, K. Kanclerz, A. Kocoń, B. Koptyra, W. Mieleszczenko-Kowszewicz, P. Miłkowski, M. Oleksy, M. Piasecki, Łukasz Radliński, K. Wojtasik, S. Woźniak, and P. Kazienko, "Chatgpt: Jack of all trades, master of none," *Information Fusion*, vol. 99, p. 101861, 2023.

[50] T. Chakrabarty, D. Ghosh, A. Poliak, and S. Muresan, "Figurative language in recognizing textual entailment," in *Findings of the ACL: ACL-IJCNLP 2021*. Online: ACL, Aug. 2021, pp. 3354–3361.

[51] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *ArXiv*, vol. abs/1909.11942, 2019.

[52] H. Batra, N. S. Punn, S. K. Sonbhadra, and S. Agarwal, "Bert-based sentiment analysis: A software engineering perspective," in *Database and Expert Systems Applications: 32nd International Conference, DEXA 2021*. Berlin, Heidelberg: Springer-Verlag, 2021, p. 138–148.

[53] R. Mao, Q. Liu, K. He, W. Li, and E. Cambria, "The biases of pre-trained language models: An empirical study on prompt-based sentiment analysis and emotion detection," *IEEE Trans. Affect. Comput.*, vol. 14, no. 3, pp. 1743–1753, 2023.

[54] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "CodeBERT: A pre-trained model for programming and natural languages," in *Findings of the ACL: EMNLP 2020*. Online: ACL, Nov. 2020, pp. 1536–1547.

[55] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," in *Proceedings of the 2019 Conference on EMNLP-IJCNLP*. Hong Kong, China: ACL, Nov. 2019, pp. 3982–3992.

[56] M. M. Imran, A. Ciborowska, and K. Damevski, "Automatically selecting follow-up questions for deficient bug reports," in *18th IEEE/ACM International Conference on Mining Software Repositories*. Madrid, Spain: IEEE, 2021, pp. 167–178.

[57] K. Ethayarajh, "How contextual are contextualized word representations? Comparing the geometry of BERT, ELMo, and GPT-2 embeddings," in *Proceedings of the 2019 Conference on EMNLP-IJCNLP*. Hong Kong, China: ACL, Nov. 2019, pp. 55–65.

[58] H. Yan, L. Gui, W. Li, and Y. He, "Addressing token uniformity in transformers via singular value transformation," in *Proceedings of the 38th Conference on Uncertainty in Artificial Intelligence*, vol. 180. Eindhoven, The Netherlands: PMLR, 2022, pp. 2181–2191.

[59] L. B. Godfrey and M. S. Gashler, "A continuum among logarithmic, linear, and exponential functions, and its potential to improve generalization in neural networks," in *2015 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K)*. USA: IEEE Computer Society, nov 2015, pp. 481–486.

[60] W. H. Gomaa, A. A. Fahmy *et al.*, "A survey of text similarity approaches," *international journal of Computer Applications*, vol. 68, no. 13, pp. 13–18, 2013.

[61] N. Cliff, "Dominance statistics: Ordinal analyses to answer ordinal questions." *Psychological Bulletin*, vol. 114, pp. 494–509, 1993.

[62] S. Mohammad, E. Shutova, and P. Turney, "Metaphor as a medium for emotion: An empirical study," in *Proceedings of the Fifth Joint Conference on Lexical and Computational Semantics*. Germany: ACL, Aug. 2016, pp. 23–33.

[63] D. Graziotin, X. Wang, and P. Abrahamsson, "Do feelings matter? on the correlation of affects and the self-assessed productivity in software engineering," *J. Softw. Evol. Process*, vol. 27, no. 7, p. 467–487, jul 2015.

[64] N. Raman, M. Cao, Y. Tsvetkov, C. Kästner, and B. Vasilescu, "Stress and burnout in open source: Toward finding, understanding, and mitigating unhealthy interactions," in *Proceedings of the ACM/IEEE 42nd ICSE: NIER*. USA: ACM, 2020, p. 57–60.

[65] D. Graziotin, F. Fagerholm, X. Wang, and P. Abrahamsson, "On the unhappiness of software developers," in *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*. USA: ACM, 2017, p. 324–333.

[66] 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 MSR*. USA: ACM, 2018, p. 364–375.

[67] C. Sun, X. Qiu, Y. Xu, and X. Huang, "How to fine-tune bert for text classification?" in *Chinese Computational Linguistics*. Cham: Springer International Publishing, 2019, pp. 194–206.

[68] Z. Ke, H. Xu, and B. Liu, "Adapting BERT for continual learning of a sequence of aspect sentiment classification tasks," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Online: ACL, Jun. 2021, pp. 4746–4755.

[69] A. Chiorrini, C. Diamantini, A. Mircoli, and D. Potena, "Emotion and sentiment analysis of tweets using BERT," in *Proceedings of the Workshops of the EDBT/ICDT 2021 Joint Conference*, ser. CEUR Workshop Proceedings, vol. 2841. Nicosia, Cyprus: CEUR-WS.org, 2021.

[70] E. Liu, C. Cui, K. Zheng, and G. Neubig, "Testing the ability of language models to interpret figurative language," in *Proceedings of the 2022 Conference of the NAACL: Human Language Technologies*. Seattle, United States: ACL, Jul. 2022, pp. 4437–4452.

[71] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on*

machine learning. PMLR, 2020, pp. 1597–1607.
[72] I. Ferreira, B. Adams, and J. Cheng, "How heated is it? understanding github locked issues," in *2022 IEEE/ACM 19th International Conference on Mining Software Repositories*. USA: IEEE Computer Society, may 2022, pp. 309–320.
[73] Z. Botev and A. Ridder, "Variance reduction," *Wiley statsRef: Statistics reference online*, pp. 1–6, 2017.
[74] N. S. Suhaimi, Z. Othman, and M. R. Yaakub, "Comparative analysis between macro and micro-accuracy in imbalance dataset for movie review classification," in *Proceedings of Seventh International Congress on Information and Communication Technology*. Singapore: Springer Nature Singapore, 2023, pp. 83–93.
[75] Y. Tian, D. Lo, and C. Sun, "Drone: Predicting priority of reported bugs by multi-factor analysis," in *2013 IEEE International Conference on Software Maintenance (ICSM)*. USA: IEEE Computer Society, sep 2013, pp. 200–209.
[76] J. Dominic, C. Ritter, and P. Rodeghero, "Onboarding bot for newcomers to software engineering," in *2020 IEEE/ACM International Conference on Software and System Processes (ICSSP)*. USA: IEEE Computer Society, jun 2020, pp. 91–94.
[77] I. d. F. Junior, S. Marczak, R. Santos, C. Rodrigues, and H. Moura, "C2m: a maturity model for the evaluation of communication in distributed software development," *Empirical Software Engineering*, vol. 27, no. 7, p. 188, 2022.
[78] R. Joseph, T. Liu, A. B. Ng, S. See, and S. Rai, "NewsMet : A 'do it all' dataset of contemporary metaphors in news headlines," in *Findings of the Association for Computational Linguistics: ACL 2023*. Toronto, Canada: ACL, Jul. 2023, pp. 10 090–10 104.
[79] K. Hilton, A. Siami Namin, and K. S. Jones, "Metaphor identification in cybersecurity texts: a lightweight linguistic approach," *SN Applied Sciences*, vol. 4, no. 2, p. 60, 2022.
[80] U. Naseem, J. Kim, M. Khushi, and A. G. Dunn, "Robust identification of figurative language in personal health mentions on twitter," *IEEE Transactions on Artificial Intelligence*, vol. 4, no. 2, pp. 362–372, 2022.
[81] T. Wijesiriwardene, A. Sheth, V. L. Shalin, and A. Das, "Why do we need neurosymbolic ai to model pragmatic analogies?" *IEEE Intelligent Systems*, vol. 38, no. 5, pp. 12–16, 2023.
[82] S. R. Fussell and M. M. Moss, "Figurative language in emotional communication," in *Social and cognitive approaches to interpersonal communication*. Psychology Press, 2014, pp. 113–141.
[83] A. Esmaeilzadeh and K. Taghva, "Text classification using neural network language model (nnlm) and bert: An empirical comparison," in *Intelligent Systems and Applications*, K. Arai, Ed. Cham: Springer International Publishing, 2022, pp. 175–189.
[84] Z. Liu, S.-h. Lei, Y.-l. Guo, and Z.-a. Zhou, "The interaction effect of online review language style and product type on consumers' purchase intentions," *Palgrave Communications*, vol. 6, no. 1, p. 11, 2020.
[85] A. Kronrod and S. Danziger, ""Wii Will Rock You!" The Use and Effect of Figurative Language in Consumer Reviews of Hedonic and Utilitarian Consumption," *Journal of Consumer Research*, vol. 40, no. 4, pp. 726–739, 07 2013.
[86] D. R. Recupero, M. Alam, D. Buscaldi, A. Grezka, and F. Tavazoee, "Frame-based detection of figurative language in tweets [application notes]," *IEEE Comput. Intell. Mag.*, vol. 14, no. 4, pp. 77–88, 2019.
[87] L. Weitzel, R. C. Prati, and R. F. Aguiar, *The Comprehension of Figurative Language: What Is the Influence of Irony and Sarcasm on NLP Techniques?* Cham: Springer International Publishing, 2016, pp. 49–74.
[88] E. Shutova, "Models of metaphor in nlp," in *Proceedings of the 48th Annual Meeting of the ACL*. USA: ACL, 2010, p. 688–697.
[89] Y. Hao and T. Veale, "An ironic fist in a velvet glove: Creative mis-representation in the construction of ironic similes," *Minds and Machines*, vol. 20, pp. 635–650, 2010.
[90] J. Peng and A. Feldman, "Automatic idiom recognition with word embeddings," in *Information Management and Big Data*. Cham: Springer International Publishing, 2017, pp. 17–29.
[91] M. S. McGlone, "Conceptual metaphors and figurative language interpretation: Food for thought?" *Journal of Memory and Language*, vol. 35, no. 4, pp. 544–565, 1996.
[92] T. Veale and Y. Hao, "A fluid knowledge representation for understanding and generating creative metaphors," in *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*. Manchester, UK: Coling 2008 Organizing Committee, Aug. 2008, pp. 945–952.
[93] J. H. Martin, "A corpus-based analysis of context effects on metaphor comprehension," *Trends in Linguistics Studies and Monographs*, vol. 171, p. 214, 2006.
[94] B. Liu, *Sentiment Analysis - Mining Opinions, Sentiments, and Emotions*. Cambridge, United Kingdom: Cambridge University Press, 2015.
[95] M. Munezero, C. Montero, E. Sutinen, and J. Pajunen, "Are they different? affect, feeling, emotion, sentiment, and opinion detection in text," *IEEE Transactions on Affective Computing*, vol. 5, no. 02, pp. 101–111, apr 2014.
[96] P. Chatterjee, K. Damevski, N. A. Kraft, and L. Pollock, "Automatically identifying the quality of developer chats for post hoc use," *ACM TOSEM*, vol. 30, no. 4, jul 2021.
[97] N. Novielli, F. Calefato, and F. Lanubile, "A gold standard for emotion annotation in stack overflow," in *Proceedings of the 15th International Conference on Mining Software Repositories*. USA: ACM, 2018, p. 14–17.
[98] A. Sajadi, K. Damevski, and P. Chatterjee, "Interpersonal trust in oss: Exploring dimensions of trust in github pull requests," in *2023 IEEE/ACM 45th International Conference on Software Engineering: NIER*. USA: IEEE Computer Society, may 2023, pp. 19–24.
[99] Z. Chen, Y. Cao, X. Lu, Q. Mei, and X. Liu, "Sentimoji: An emoji-powered learning approach for sentiment analysis in software engineering," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. USA: ACM, 2019, p. 841–852.
[100] M. R. Islam and M. F. Zibran, "Leveraging automated sentiment analysis in software engineering," in *Proceedings of the 14th International Conference on Mining Software Repositories*, J. M. González-Barahona, A. Hindle, and L. Tan, Eds. Buenos Aires, Argentina: IEEE Computer Society, 2017, pp. 203–214.
[101] I. Ferreira, A. Rafiq, and J. Cheng, "Incivility detection in open source code review and issue discussions," *Journal of Systems and Software*, p. 111935, 2023.
[102] F. A. Acheampong, H. Nunoo-Mensah, and W. Chen, "Transformer models for text-based emotion detection: A review of bert-based approaches," *Artif. Intell. Rev.*, vol. 54, no. 8, p. 5789–5829, dec 2021.
[103] J. Sarker, "Identification and mitigation of toxic communications among open source software developers," in *Proceedings of the 37th ICSE*. USA: ACM, 2023.
[104] O. Chaparro, J. Florez, U. Singh, and A. Marcus, "Reformulating queries for duplicate bug report detection," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. USA: IEEE Computer Society, feb 2019, pp. 218–229.
[105] O. Chaparro, J. Lu, F. Zampetti, L. Moreno, M. Di Penta, A. Marcus, G. Bavota, and V. Ng, "Detecting missing information in bug descriptions," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. USA: ACM, 2017, p. 396–407.
[106] L. A. F. Gomes, R. da Silva Torres, and M. L. Côrtes, "Bug report severity level prediction in open source software: A survey and research opportunities," *Information and Software Technology*, vol. 115, pp. 58–78, 2019.