

How Effectively Do LLMs Extract Feature-Sentiment Pairs from App Reviews?

Faiz Ali Shah*, Ahmed Sabir*, Rajesh Sharma, and Dietmar Pfahl
University of Tartu, Institute of Computer Science, Tartu, Estonia

Abstract—Automatic analysis of user reviews to understand user sentiments toward app functionality (*i.e.* app features) helps align development efforts with user expectations and needs. Recent advances in Large Language Models (LLMs) such as ChatGPT have shown impressive performance on several new tasks without updating the model's parameters *i.e.* using zero or a few labeled examples, but the capabilities of LLMs are yet unexplored for feature-specific sentiment analysis. The goal of our study is to explore the capabilities of LLMs to perform feature-specific sentiment analysis of user reviews. This study compares the performance of state-of-the-art LLMs, including GPT-4, ChatGPT, and different variants of Llama-2 chat, against previous approaches for extracting app features and associated sentiments in zero-shot, 1-shot, and 5-shot scenarios. The results indicate that GPT-4 outperforms the rule-based SAFE by 17% in f1-score for extracting app features in the zero-shot scenario, with 5-shot further improving it by 6%. However, the fine-tuned RE-BERT exceeds GPT-4 by 6% in f1-score. For predicting positive and neutral sentiments, GPT-4 achieves f1-scores of 76% and 45% in the zero-shot setting, which improve by 7% and 23% in the 5-shot setting, respectively. Our study conducts a thorough evaluation of both proprietary and open-source LLMs to provide an objective assessment of their performance in extracting feature-sentiment pairs.¹

Index Terms—App Feature Extraction, Large Large Models, Sentiment Prediction, Software Maintenance, Software Evolution

1 INTRODUCTION

Mobile app users provide feedback on the app's functionality, usage scenarios, and desired features by submitting reviews through app marketplaces [1], [2]. Analyzing this feedback can help developers understand users' perceptions of app features and their evolving needs [3]. Due to the large amount of daily feedback, manual analysis of user reviews is impractical. Several techniques have been proposed to automatically summarize user feedback to facilitate software engineering activities, such as software maintenance and evolution [4], [5].

A dedicated theme within the field of opinion mining [1] focuses on feature-specific sentiment analysis (also called aspect-based sentiment analysis), aiming to automatically generate summaries of users' sentiments towards the functionality of software applications, commonly referred to as app features.² The task of feature-specific sentiment analysis of user reviews can be decomposed into two sub-tasks. The first task, "app feature extraction", is to identify the exact words

expressing the functional features³ of an app in a review text, and the second task, "feature-specific sentiment prediction", involves determining the sentiment (*i.e.* positive, negative, or neutral) conveyed toward the features. For instance, in a review sentence, *it is great for taking down notes*, the output of feature-specific sentiment analysis would be a feature-sentiment pair such as ("taking down notes", "positive").

Various methods have been proposed to identify app features and associated sentiments from app reviews automatically. Guzman's method [3] involved identifying collocations as potential app features and then analyzing their associated sentiments using the SentiStrength⁴ tool. Dragoni's approach [6] uses linguistic rules based on part-of-speech patterns and Semantic Dependency Graph (SDG) to extract app features and corresponding opinion words from user reviews. Their approach relies on lexical dictionaries to determine the sentiment polarity of opinion words. Some studies have only focused on the task of extracting app features. For this purpose, rule-based approaches like SAFE [7] or supervised methods like RE-BERT [8] and T-FREX [9] have been proposed. The results of these studies indicate that fine-tuning of pre-trained models such as BERT has significantly outperformed rule-based approaches such as SAFE for extracting app features. However, to fine-tune such models, a considerable number of task-specific training examples are still required. Furthermore, updating some of the model parameters to align with the task adds more complexity to the model fine-tuning.

Recently, Large Language Models (LLMs) trained on a large corpus of data with Reinforcement Learning from Human Feedback (RLHF), such as ChatGPT [10] and Llama-2 [11], have shown the ability to generalize to new tasks without requiring task-specific fine-tuning [12]. LLMs take in natural language instructions defining the task (labeled as A in Figure 1), optionally accompanied by a few examples for task demonstration (labeled as B in Figure 1). This learning method is called *zero-shot* when no examples are provided with the instruction, and *few-shot* when a few examples are supplied with the task description. LLMs with RLHF have been proven effective in following instructions for different tasks with *zero-shot* or *few-shot* learning, eliminating the need for task-specific training examples for model parameter updating.

*Equal contribution.

¹<https://github.com/Faiz-UT/Eval-Feature-Sentiment-Extraction-LLMs>

²Our study used the terms "feature" and "app features" interchangeably

³Similar to the study of [2], we don't consider non-functional attributes of an app as features.

⁴<https://github.com/zhunhung/Python-SentiStrength>

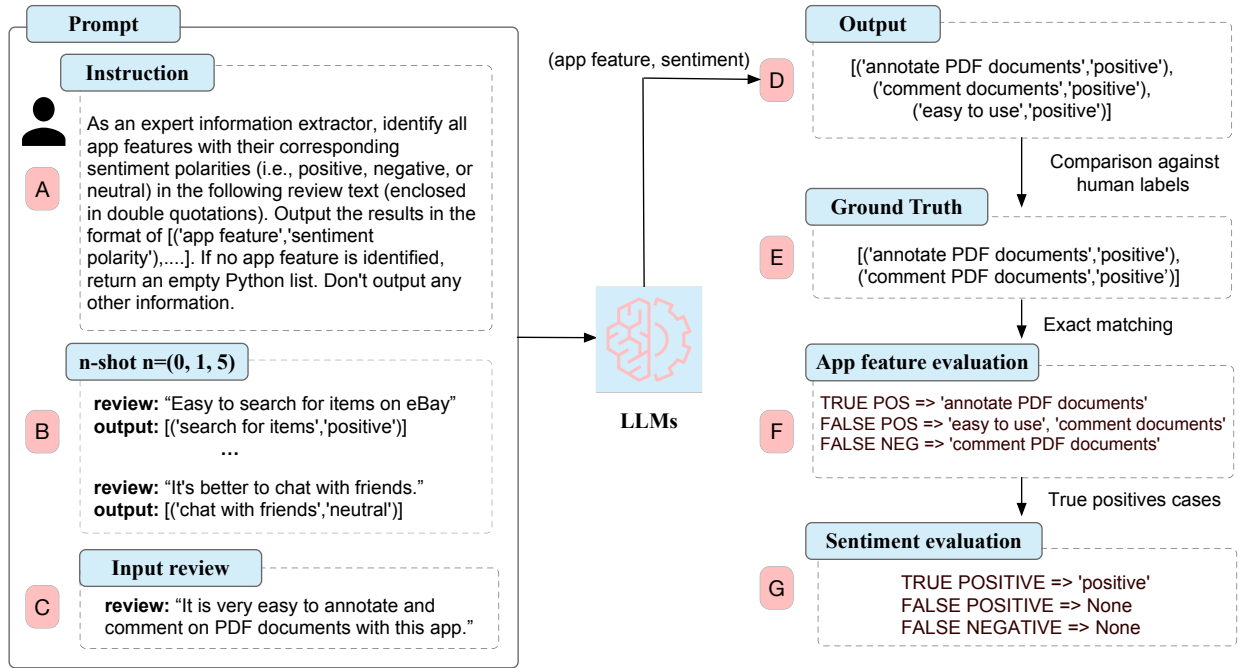


Fig. 1: Our approach for evaluating *zero-shot* and *few-shot* capabilities of LLMs for extracting (app feature, sentiment) pairs from a user review.

Zhang conducted a study [13] that shows ChatGPT is highly effective in automatically extracting app features from app descriptions, even in *zero-shot* settings. Nevertheless, app reviews can be more challenging to extract app features due to informal language, noise, and a lot of developer-irrelevant information [14]. Consequently, it remains unexplored how effectively LLMs can simultaneously extract app features and feature-specific sentiments, as demonstrated in Figure 1. To this end, this study aims to investigate the effectiveness of LLMs for extracting features and corresponding sentiments from app reviews under *zero-shot* and *few-shot* settings by answering the following research questions:

RQ1: How does the *zero-shot* performance of LLMs compare to existing methods for extracting feature-sentiment pairs from app reviews?

To address RQ1, under the *zero-shot* scenario, we prompted state-of-the-art LLMs fine-tuned with RLHF, including GPT-4, ChatGPT, and various sizes of LLaMA-2 chat models (7B, 13B, and 70B) for extracting feature-sentiment pairs from user reviews. To evaluate the performance of LLM models, LLM predicted features and associated sentiments (labeled as 'D' in Figure 1) are compared against human-labeled feature-sentiment pairs (labeled as 'E') using exact and partial feature matching strategies. Our results show that all LLM models have outperformed the previous rule-based approaches for extracting app features from app reviews when employing the exact match strategy. GPT-4 delivered the most impressive results, showing a substantial 23.6% improvement in the f1-score over the best-performing rule-based approach. Regarding feature-specific sentiment prediction, GPT-4 emerged as the top-performing model, achieving an f1-score of 74% for

predicting the *positive* sentiment. LLaMA-2-70B demonstrated the best f1-score of 50.4% for predicting the *negative* feature sentiment. For the *neutral* sentiment prediction, GPT-4 outperformed other models with the best f1-score of 41%.

RQ2: How does the *few-shot* performance of LLMs compare to *zero-shot* and existing methods for extracting feature-sentiment pairs from app reviews?

To address RQ2, we conducted an evaluation using the same LLMs as those employed in addressing RQ1, but under *1-shot* and *5-shot* scenarios. Our findings indicate that, compared to the *zero-shot* performance, in terms of app feature extraction, the *5-shot* approach led to an improvement in the f1 performance of LLaMA-2-70B, ChatGPT, and GPT-4 by 10.6%, 3.5%, and 6%, respectively, when employing the partial match strategy with a 2-word difference. Especially, GPT-4 attained an f1-score of 56% with the *5-shot* approach, marking a significant 23% enhancement over the SAFE approach. Furthermore, in predicting the *positive* feature-specific sentiment, the *5-shot* approach resulted in a 7% increase in the f1-score for GPT-4 and a 3% improvement for LLaMA-2-70B compared to their *zero-shot* performances. For predicting *neutral* sentiment, GPT-4 exhibited a significant gain of 23%.

The paper is structured as follows: Section 2 offers background knowledge on LLMs. Section 3 summarizes related work concerning feature-specific sentiment analysis of app reviews. The details pertaining to the labeled dataset, LLM models, and their evaluation procedure are elaborated in Section 4. The outcomes of these evaluations are presented in Section 5, followed by a discussion in Section 6. Section 7 addresses the limitations and potential threats to the validity of this study. Finally, the paper concludes with a presentation

of the conclusions and future avenues for research in Section 8.

2 BACKGROUND

Large Language Models (LLMs) have experienced a remarkable surge in popularity and attention from both the general public and Natural Language Processing practitioners. These models have demonstrated significant advancements beyond the current state-of-the-art in numerous natural language tasks. These remarkable improvements led to the growing discussion of adopting such models in many everyday tasks, such as offering medical guidance, organizing job-related and a range of other applications [12].

LLMs have become essential in natural language processing primarily due to their ability to enhance data efficiency for targeted tasks. Fine-tuning the model parameters using gradient-based techniques for specific downstream tasks is critical to better performance. Although the fine-tuning method has yielded better performance, high-quality human-annotated label data is required.

Another alternative method that requires no fine-tuning is In-Context Learning (ICL) [15], [16]. ICL enables LLMs to perform unseen tasks without any training by feeding a small number of examples as part of the input.

No Demonstration is a *zero-shot* prompting that refers to the ability of the LLMs model to perform a task without any prior knowledge related to that specific task.

Demonstration w/ Label *One-shot* prompting is an ICL-based demonstration with a single label that involves providing a specific example input along with a corresponding output. The task is to guide the LLMs model to generate text relevant to the given task.

Demonstration w/ Labels *Few-Shot* prompting is a similar approach to *one-shot* ICL-based demonstration; however, the model needs to be provided with a collection of examples (*e.g.* *3-shot*, *5-shot*, *etc.*) and a single unlabeled example, which needs to be predicted by the model.

3 RELATED WORK

In this section, we discuss the previous research on feature-specific sentiment analysis of app reviews. We categorized the work into rule-based methods, supervised learning/fine-tuned language models (LMs), and Instruction-aligned human feedback (*i.e.* RLHF) based Large Language Models (LLMs).

3.1 Rule-based methods

Rule-based methods use predefined patterns to extract information based on linguistic patterns or regular expressions from review text. Dragoni *et al.* [6] exploits linguistic rules consisting of part-of-speech patterns and semantic dependency relations to extract app features and their associated opinion words from a review simultaneously. To determine the sentiment polarity (*i.e.* positive, neutral or negative), their approach relies on lexical dictionaries. Guzman *et al.* [3] proposed a method identifying collocations of nouns, verbs, and adjectives as app features from app reviews, and then SentiStrength⁵ tool

is applied to determine users' sentiment towards the extracted app features. The same approach proposed by Guzman is used by Dalpiaz *et al.* [4] and Shah *et al.* [5] for performing feature-level sentiment analysis but for comparing competing apps.

SUR-Miner [17] initially employed pre-defined templates derived from the Semantic Dependency Graph (SDG) to extract feature-opinion pairs from a review sentence. Subsequently, it utilizes the sentiment analysis tool⁶ "Deeply moving" to determine user's sentiments towards features based on opinion words.

The SAFE method is introduced by Johann *et al.* [7] utilizing 18 part-of-speech and 5 sentence patterns to extract app features from user reviews. When conducting feature-specific sentiment analysis on user reviews, the SAFE method is capable only of extracting app features. Hence, an additional step is required to ascertain the sentiments of users towards the SAFE-extracted features as demonstrated in the work of [18]–[20].

In contrast to previous studies that focused on identifying specific app features and their associated sentiments, MARK framework [21] introduced a keyword-based approach for discovering and understanding users' opinions. Tushev's work [22] presented a keyword-assisted topic modeling approach for generating coherent topics. A method called "Casper" [23] is introduced to identify more general app issues known as "mini user stories" from review texts using parts-of-speech tagging and dependency parsing patterns. A new technique is introduced by Bakar *et al.* [24] for extracting software features as phrases from user reviews to enable reusing natural language requirements.

The rule-based approaches mentioned above are limited in their ability to identify complex patterns, domain-specific terminology, or grasp context. These limitations hinder their generalization capabilities when applied to feature-level sentiment analysis. For example, both Dąbrowski *et al.* [25] and Shah *et al.* [26] discovered that the SAFE approach exhibited lower precision and recall on additional datasets compared to the results reported in the original study.

3.2 Supervised Learning

Supervised learning aims to train a machine learning model to learn the mapping between input features and output labels based on the labeled training data. Wang's work [27] treated the task of app feature extraction as Named Entity Recognition [28] (NER) task and built a supervised CRF model for identifying problematic features of apps from user reviews. Their results demonstrate that the supervised CRF model outperforms pattern-based feature-extraction techniques such as Caspar and SAFE.

Fine-tuning LLMs involves taking a pre-trained model such as BERT or RoBERTa that has already been trained on a large corpus and then updating its parameters on the smaller task-specific dataset to adapt it to perform a new or more specialized task. RE-BERT [8], a fine-tuned model, is introduced

⁵<http://sentistrength.wlv.ac.uk/>

⁶<https://nlp.stanford.edu/sentiment/>

that extracts software requirements from app reviews. REBERT [8] outperformed previous rule-based approaches such as SAFE in terms of performance with a mean score of 46% using an exact matching strategy. A new approach named [29] "KEFE" is offered to identify key features from app reviews. They used SAFE rules to obtain candidate phrases and then employed a BERT-based model to classify each phrase as feature-describing or non-feature-describing phrase. Recently, a study [9] was conducted that fine-tuned transformer-based models such as BERT, RoBERTa, and XLNet on labeled app reviews for extracting app features, and evaluated the performance of these models under in-domain and out-of-domain settings.

These studies suggested that fine-tuning pre-trained language models is better for identifying app features from user reviews than rule-based methods. However, such techniques require training examples for fine-tuning the model, which can be expensive to obtain. Furthermore, these models are not trained to simultaneously extract feature and sentiment pairs from user reviews.

3.3 LLMs with RLHF

LLMs are often used to follow instructions (prompt) to execute the user's tasks. However, quite often, these models generate less explicit intentions than following instructions. To address this challenge, Ouyang *et al.* [30] propose InstructGPT, a method of fine-tuning a pre-trained language model through Reinforcement Learning from Human Feedback (RLHF). Specifically, the InstructGPT training starts with supervised learning based on a dataset of human-written prompts and responses. This initial phase is designed to teach the model the basic structure of understanding and generating language in a way that is aligned with human expectations. In Reinforcement Learning (RL) stage, the model receives feedback from humans to fine-tune its responses with a reward, that acts as a guide for its adjustments, ensuring they are both relevant and coherent. This iterative process enhances the model's ability to understand and meet the intricate demands of human communication.

LLMs with RLHF have demonstrated remarkable performance in a variety of downstream tasks without the need for parameter tuning. These models have shown their ability to perform various tasks by demonstrating a few examples *few-shot* or by describing the task through instructions alone *zero-shot* [15]. Recent models (*e.g.* ChatGPT [10], LLAMA-2 Chat [11], *etc.*) build upon the foundations laid by InstructGPT, resulting in substantial improvements over LLMs trained solely on text corpora through unsupervised pre-training.

In a recent study [13], the performance of ChatGPT is evaluated to extract app features from app descriptions under *zero-shot* settings. The results showed that the model achieved an 84% precision and a 75% recall rate on extracting app features from app descriptions. Building upon the encouraging results of this study [13], our study aims to evaluate the performance of RLHF chat-based LLMs in extracting feature-

TABLE 1: Statistics of the labeled review dataset.

App	Reviews		Feature-Sentiments			Total
	reviews	sents	positive	neutral	negative	feature&sentiment
Evernote	125	367	97	189	9	295
Facebook	125	327	49	168	25	242
eBay	125	294	95	102	9	206
Netflix	125	341	79	159	24	262
Spotify	125	341	32	122	26	180
Photo editor	125	154	39	47	10	96
Twitter	125	183	5	93	24	122
WhatsApp	125	169	20	84	14	118
Overall	1000	2062	416	964	141	1521

sentiment pairs from user reviews under *zero-shot* and *few-shot* settings.

4 EXPERIMENTAL SETTINGS

This section first describes the labeled dataset utilized to assess the performance of LLMs. Next, we provide an overview of the LLMs that have been evaluated, along with the benchmarks used for performance comparison. Finally, we describe the methodology employed to measure the performance of LLMs and elaborate on the implementation details.

4.1 Labeled dataset

The review dataset curated by Dąbrowski *et al.* [25], which is utilized to answer Research Questions 1 and 2 (RQ1 and RQ2), is previously used to evaluate the performances of rule-based approaches in extracting feature-sentiment pairs. Two human annotators labeled 1000 user reviews for eight different applications. The dataset comprises 1521 labeled pairs associating features with corresponding sentiments. Table 1 summarizes the labeled reviews and feature-sentiment statistics within this dataset.

As we can see in Table 1, the app *EverNote* mentions the highest number of app features, while the app *Photo Editor* mentions the lowest number of app features. The sentiment distribution is imbalanced in the dataset, and most features have a sentiment polarity of *neutral*. The number of features with a sentiment polarity of *negative* is the least.

4.2 Experimented LLMs

We leverage the most recent state-of-the-art RLHF-based LLM models that are known for their ability to generate coherent and contextually relevant text:

ChatGPT [10] & GPT-4 [31]. follows InstructGPT [30] training procedure via RLHF, where the model is initially adjusted using a collection of human-generated text responses. This step aims to establish a preliminary understanding and mimicry of human conversational patterns.

LLAMA-2 Chat [11]. is an open-source RLHF-based LLM model that is trained on a new mix of publicly available data. Llama 2 Chat is a LLAMA-2 version that is optimized for dialogue use cases. The model comes in different sizes (7B, 13B, and 70B).

4.3 Baselines

We compared the performance of our LLM-based models with four state-of-the-art baseline methods briefly described in this section. Since Dąbrowski et al.’s study [25] evaluated these baseline methods using the same labeled dataset and evaluation procedure as our study, we used their performance results rather than re-running the baseline methods ourselves. **GuMA**⁷ [3], extracts app features and then determines the sentiments towards the extracted features. Both feature extraction and sentiment prediction tasks are performed independently of each other. To identify app features, GuMA uses collocations-finding algorithms that find a combination of words that co-occur frequently in the review text. For predicting sentiments, this approach first applied the SentiStrength⁸ tool that assigns a sentiment polarity score to a review sentence, and then the sentiment score is computed for the features.

SAFE [7], identifies app features using 18 Part-of-Speech (POS) patterns and five sentence patterns from user reviews. Since it can only extract app features in a review, an additional sentiment analysis tool needs to be applied to determine sentiment polarity towards the features.

ReUS [6], uses a set of linguistic rules derived from part-of-speech patterns and semantic dependency relations to detect app features and their corresponding opinion words from a user review. Different from GuMA, REUS performed both tasks at the same time. To determine the sentiment polarity, the ReUS approach relies on lexical dictionaries.

RE-BERT [8], posed the problem of app feature extraction from app reviews as a token classification task. This approach fine-tuned the pre-trained BERT model on the review dataset described in Section 4.1 to extract app features. RE-BERT achieved better performance for app feature extraction than the rule-based approaches GUMa, SAFE, and ReUs. Since RE-BERT doesn’t extract the sentiments conveyed towards the extracted features, a separate step is required to perform feature-specific sentiment analysis.

4.4 Evaluation procedure

Since the app feature annotations can often be noisy and ambiguous, similar to previous studies [8], [25], [32], we adopted a token-based evaluation method using exact and partial matching (by difference of 1- word and 2-word) between predicted and human-annotated features.

Token-based matching counts and evaluates each instance of an app feature separately. For instance, if an app feature *voice message* occurs several times in different reviews, each feature instance will be counted separately. It might happen that depending on the context, not all *voice message* bi-grams are annotated as app features. This approach distinguishes between app features and non-app features expressed with the same sequence of words.

Exact matching requires the predicted and annotated app features to match exactly. For instance, if the annotated feature

is *send my message*, then to count it as a match, the predicted app feature must consist of exactly the same tokens. If the feature extraction approach extracts *send message* as a feature, omitting the possessive pronoun *my*, the prediction is counted as a false positive under the exact match strategy.

Partial matching allows some mismatch when comparing the predicted app features with the human-annotated features. In a partial match strategy that permits the difference of one word, the predicted feature *send message* will be considered true positive even though the human-annotated app feature is *send my message*. However, the predicted feature *message* would be counted as a false positive because it differs from the annotated app feature in more than one word. Similarly, a partial match strategy allows the difference of two words; a predicted feature *to send my message* would be counted as true positive, but a longer predicted feature such as "failed to send my message" is counted as false positive.

For evaluating the feature extraction performance of LLMs, we computed the precision, recall, and f1-score performance of LLMs (see Section 4.2) using exact and partial matching strategies on the whole dataset. For evaluating the sentiment prediction performance, same as the study of Dąbrowski et al. [25], we calculated the precision, recall, and f1-score for each sentiment polarity (*i.e.* positive, neutral, negative) on true positive features using exact and partial matching strategies.

4.5 Implementation Details

For *zero-shot* experiments, we prompted LLM models with short and long prompts without demonstrating any labeled examples. The *short prompt* directly instructs the model to perform the task of feature-sentiment pair extraction from an input review. On the other hand, the *long prompt* elaborates on key concepts such as feature, feature expression, and sentiment polarity relevant to the task of feature-extraction pair, and then instructs the LLM model to proceed with the task. In this study, we denote the short prompt as *S-prompt* and the long prompt as *L-prompt*.

To assess the performance of LLMs under *few-shot* conditions, we conducted two experiments. In the first experiment, a single labeled example containing true features and sentiment labels was incorporated into the prompt (*1-shot*) to guide the model in understanding the task. Subsequently, in the second experiment (*5-shot*), five labeled reviews were included in the prompt to provide the models’ richer contextual knowledge of the task. In both the *1-shot* and *5-shot* experiments, labeled examples were randomly selected from a pool of 10 review samples during each inference. These labeled examples were taken from the annotation guidelines⁹ prepared by Dąbrowski et al. [25].

Our prompts instruct LLM models to provide feature-sentiment pairs as a Python list of tuples or return an empty Python list if no feature is found in the input review. Nevertheless, the LLM model can produce text that either includes a Python list with additional information or excludes the

⁷Same as [2], we refer to the approach using abbreviations derived from authors’ surnames

⁸<http://sentistrength.wlv.ac.uk/>

⁹<https://github.com/jsdabrowski/IS-22>

TABLE 2: Comparison of *zero-shot* performances of LLMs for extracting feature-sentiment pairs with S-prompt and L-prompt, and using exact and partial match strategies. (Second best result and Third best result). The (*) refers to the model fine-tuned on the dataset.

Model	Prompt type	Exact match (n = 0)			Partial match 1(n = 1)			Partial match 2(n = 2)		
		Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
GuMa [3]	-	0.05	0.13	0.07	0.15	0.37	0.21	0.18	0.44	0.25
SAFE [7]	-	0.06	0.06	0.06	0.24	0.24	0.24	0.33	0.34	0.33
ReUS [6]	-	0.08	0.08	0.08	0.18	0.18	0.18	0.33	0.25	0.25
RE-BERT* [8]	-	-	-	0.46	-	-	0.55	-	-	0.62
ChatGPT [10]	S	0.227 ± 0.03	0.406 ± 0.04	0.290 ± 0.04	0.294 ± 0.05	0.527 ± 0.06	0.377 ± 0.05	0.346 ± 0.04	0.620 ± 0.04	0.443 ± 0.04
ChatGPT	L	0.219 ± 0.04	0.433 ± 0.05	0.290 ± 0.04	0.278 ± 0.05	0.551 ± 0.07	0.369 ± 0.06	0.326 ± 0.04	0.648 ± 0.04	0.433 ± 0.04
GPT-4 [31]	S	0.257 ± 0.04	0.404 ± 0.06	0.313 ± 0.05	0.342 ± 0.06	0.537 ± 0.07	0.417 ± 0.06	0.410 ± 0.05	0.644 ± 0.06	0.500 ± 0.05
GPT-4	L	0.240 ± 0.04	0.466 ± 0.05	0.316 ± 0.05	0.313 ± 0.07	0.605 ± 0.08	0.412 ± 0.07	0.373 ± 0.06	0.723 ± 0.06	0.491 ± 0.06
Llama-2-7B Chat [11]	S	0.157 ± 0.03	0.295 ± 0.05	0.205 ± 0.03	0.211 ± 0.03	0.396 ± 0.06	0.275 ± 0.04	0.255 ± 0.03	0.479 ± 0.05	0.332 ± 0.03
Llama-2-7B Chat	L	0.124 ± 0.01	0.298 ± 0.04	0.175 ± 0.02	0.168 ± 0.02	0.403 ± 0.05	0.237 ± 0.02	0.202 ± 0.02	0.485 ± 0.04	0.285 ± 0.02
Llama-2-13B Chat	S	0.177 ± 0.04	0.265 ± 0.06	0.212 ± 0.05	0.231 ± 0.06	0.345 ± 0.07	0.277 ± 0.06	0.280 ± 0.05	0.420 ± 0.07	0.336 ± 0.06
Llama-2-13B Chat	L	0.141 ± 0.02	0.276 ± 0.04	0.187 ± 0.03	0.190 ± 0.02	0.371 ± 0.05	0.251 ± 0.03	0.231 ± 0.03	0.452 ± 0.06	0.305 ± 0.03
Llama-2-70B Chat	S	0.218 ± 0.05	0.192 ± 0.03	0.202 ± 0.03	0.286 ± 0.06	0.254 ± 0.04	0.267 ± 0.04	0.337 ± 0.05	0.300 ± 0.04	0.314 ± 0.04
Llama-2-70B Chat	L	0.248 ± 0.06	0.273 ± 0.04	0.259 ± 0.05	0.323 ± 0.06	0.357 ± 0.06	0.338 ± 0.05	0.381 ± 0.05	0.422 ± 0.05	0.399 ± 0.04

Python list altogether. Hence, we employ a regular expression to extract information formatted as a Python list from the generated text of the model. In case, the regular expression couldn't find the Python list in the output, an empty Python list is returned as an output. Upon detecting a Python list within the output generated by LLMs, our script verifies its validity as a list of tuples. Each tuple within the list is expected to contain two non-empty values in string format. Furthermore, the script validates that the second value of each tuple represents a valid sentiment, such as 'positive', 'neutral', or 'negative'. Once these conditions are satisfied, the list is considered a valid collection of predicted feature-sentiment pairs.

For all *zero-shot* and *few-shot* experiments, we executed three iterations of each LLM model on the complete labeled dataset and calculated the average performance scores – precision, recall, and f1-score – alongside their standard deviation (see Section 4.4) for both tasks: app feature extraction and feature-specific sentiment prediction.

We utilized the OpenAI Python library¹⁰ to prompt ChatGPT (gpt-3.5-turbo-0613) and GPT-4 (gpt-4-0613) models. For prompting LLAMA-2-Chat-models, we used models from huggingface¹¹ and transformers library version 4.39.2¹². Throughout all experiments, we set the temperature $\tau = 0$ to ensure that the model selects the highest probability in a greedy manner. Note that the temperature setting is a method to adjust the probability distribution used by the model to generate text. The *max_new_tokens* is set to 1000. Evaluation experiments with Llama-2-7b-chat-hf and Llama-2-13b-chat-hf were run using two NVIDIA Tesla V100 GPUs having 32 GB VRAM, while experiments with Llama-2-70b-chat-hf model were executed using four A100 GPUs with 80 GB VRAM.

5 RESULTS

This section answers our RQs by comparing the *zero-shot* and *few-shot* performance results of LLMs with baseline methods for extracting app features and corresponding sentiments from app reviews.

5.1 RQ1 - Comparison of *zero-shot* LLM performance and baseline methods for extracting feature-sentiment pairs

This section presents the results of *zero-shot* performance of LLMs for extracting app features, followed by the results on feature-specific sentiment prediction.

5.1.1 Feature extraction performance: Table 2 presents the *zero-shot* performance of LLM models for extracting app features from app reviews. The table shows the best average LLM performances in bold, highlighting the second and third-best LLM results. With the exact matching strategy, all LLM models outperformed rule-based approaches (*i.e.* GuMa, SAFE, and ReUs) in all performance measures (*i.e.* precision, recall, and f1-score). The GPT-4 has improved the average f1-score by 23.6% over the best-performing rule-based approach ReUS. However, RE-BERT, a fine-tuned model, has shown superior performance than the best-performing GPT-4 model by 14% in terms of average f1-score.

By relaxing the matching strategy by the difference of one word, all LLM models have shown an increase in performance. The highest gain of 10% in the average f1-score is yielded by the GPT-4 model with the S-prompt, and the lowest increase of 6% in the average f1-score is shown by the LLama-2-7B model with L-prompt. Except for LLama-2-7B with L-prompt, LLM models have outperformed all rule-based approaches. Again, the highest gain of 17.7% in the average f1-score is attained by GPT-4 with S-prompt over the SAFE performance (*i.e.* 24% f1-score).

Applying partial matching with the difference of two words has further boosted the precision and recall of all LLM models. The highest increase of 8% in average f1-score is yielded for

¹⁰<https://github.com/openai/openai-python>

¹¹<https://huggingface.co/meta-llama>

¹²<https://pytorch.org/project/transformers/>

the GPT-4 model with S-prompt. GPT-4 model has shown the best average f1-score of 50%, which is an improvement of 12% in the average f1-score against the best-performing rule-based approach SAFE (i.e. 33%). RE-BERT (i.e. a fine-tuned model) has again shown superior performance over the best GPT-4 model by 12% average f1-score. In our experiments, for app feature extraction, the L-prompt led to better recall but lower precision than the S-prompt across all models.

Finding 1: GPT-4 surpasses SAFE by 17% in f1-score. However, the fine-tuned RE-BERT outperforms GPT-4 by 12% in f1-score.

5.1.2 Feature-specific sentiment prediction performance:

Similar to the study of Dąbrowski *et al.* [25], we only considered true positive features obtained in RQ1 and formed three datasets, each corresponding to true positive features and corresponding sentiments from the exact match, partial match ($n = 1$), and partial match ($n = 2$).

Note that the sentiment prediction performances of LLMs are not fully comparable with ReUS and GuMa (See Section 4.3) because LLM models have detected a higher number of true positive features. As observed in RQ1 results, loosening the match criteria increases the rate of true positive features. Therefore, in Figure 2, we compared the sentiment prediction performance of LLMs with S-prompt and L-prompt using the partial match ($n = 2$), which results in the highest number of true positive features. Figure 2 shows the average sentiment prediction performances of LLMs across three runs. It is evident that performance varies depending on the sentiment polarity. The highest average f1 performance is attained for predicting *positive* sentiment, followed by the *negative* sentiment. The lowest average f1-score is observed for predicting *neutral* sentiment. The error bars in Figure 2, representing standard deviation, highlight the greater variability in LLMs for sentiment prediction performance.

As we can see in Figure 2, GPT-4 model with L-prompt achieved the highest average f1-score of 76% for predicting *positive* sentiment. Surprisingly, the LLama-2-70B model with the S-prompt has shown the best average f1-score performance for predicting *negative* sentiment. The best average f1-score for the *neutral* sentiment is 41% for GPT-4 with L-prompt. Interestingly, L-prompt improved the average f1-score performance for *neutral* sentiment across all LLM models with the exception to LLama-7b. On average, the highest improvement of 19% f1-score is yielded by LLama-2-70B, and the lowest increase of 2.4% is shown by ChatGPT.

Finding 2: To predict positive and neutral sentiments, GPT-4 achieves the best f1 scores of 76.1% and 44.8%, respectively; while LLama-2-70B yields the best f1-score of 50.4% for negative sentiment prediction.

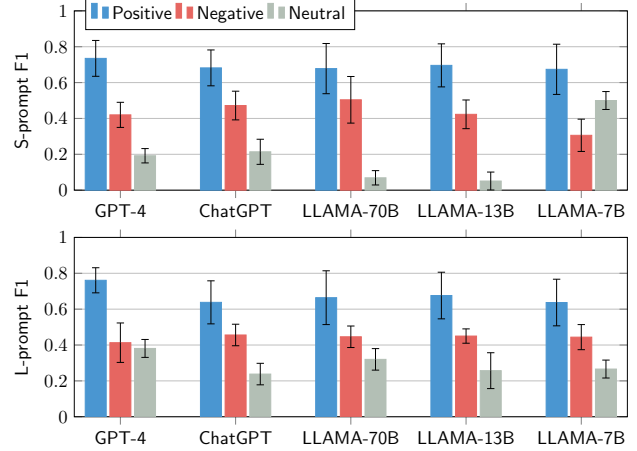


Fig. 2: Comparison of 0-shot performances of LLMs in predicting sentiment with S-prompt (upper plot) and L-prompt (lower plot) and using partial match ($n = 2$). All models, except LLama-7B, demonstrate improved performance with the L-prompt for predicting *neutral* sentiment.

5.2 RQ2 - Comparison of few-shot LLM performance against zero-shot and baseline methods for extracting feature-sentiment pairs

We compare the *few-shot* (i.e. 1-shot and 5-shot) performance of LLMs with zero-shot and baseline methods for extracting feature-sentiment pairs.

5.2.1 Feature extraction performance: Table 3 shows the LLM performances for extracting app features from user reviews under 1-shot and 5-shot scenarios. We included the zero-shot result from RQ1 for contextualization. Since *few-shot* is to help LLMs efficiently adapt to new tasks or domains with few labeled examples, we only used S-prompt for our *few-shot* experiments. Same as RQ1, we present the results of feature extraction with exact and partial match strategies.

For the exact match strategy, in comparison to the zero-shot performance, 1-shot has shown a slight improvement of almost 2% in the average f1-score of the GPT-4, LLama-7B, LLama-70B models. However, the average f1-score of the ChatGPT and LLama-13B has decreased by 2.8% and 0.3%, respectively. With the exception of ChatGPT, the demonstration of 5 labeled examples (5-shot) has increased the LLM model performances; the highest gain of 6.9% and 6.1% is shown by GPT-4 and LLama-70B, respectively. With 5-shot performance, GPT-4 is the best LLM model with an f1-score of 38.2%, which is 30% better than the SAFE approach. Nevertheless, the fine-tuned RE-BERT model is 8% better than GPT-4 in f1-score.

Compared to the zero-shot performance, for the partial match strategy, with the difference of one word, only the average f1-score of ChatGPT is dropped by 1.4%, while other models have shown an increase between 1% to 3%. The highest improvement of 3% in the average f1-score is shown by LLama-70B. The 5-shot has boosted the performance of all LLM models, the smallest increase of 1% f1-score is seen

TABLE 3: Comparison of *zero-shot*, *1-shot*, and *5-shot* performances of LLMs for extracting app features from app reviews using exact and partial match strategies. (Second best result and Third best result). The (*) refers to the model fine-tuned on the dataset.

Model	Shot	Exact match (n = 0)			Partial match 1(n = 1)			Partial match 2(n = 2)		
		Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
GuMa [3]	-	0.05	0.13	0.07	0.15	0.37	0.21	0.18	0.44	0.25
SAFE [7]	-	0.06	0.06	0.06	0.24	0.24	0.24	0.33	0.34	0.33
ReUS [6]	-	0.08	0.08	0.08	0.18	0.18	0.18	0.33	0.25	0.25
RE-BERT* [8]	-	-	-	0.46	-	-	0.55	-	-	0.62
ChatGPT [10]	0	0.227 ± 0.03	0.406 ± 0.04	0.290 ± 0.04	0.294 ± 0.05	0.527 ± 0.06	0.377 ± 0.05	0.346 ± 0.04	0.620 ± 0.04	0.443 ± 0.04
	1	0.195 ± 0.02	0.402 ± 0.03	0.262 ± 0.03	0.270 ± 0.04	0.556 ± 0.05	0.363 ± 0.05	0.323 ± 0.03	0.668 ± 0.03	0.434 ± 0.04
	5	0.210 ± 0.03	0.370 ± 0.04	0.268 ± 0.03	0.305 ± 0.04	0.536 ± 0.04	0.388 ± 0.04	0.375 ± 0.03	0.662 ± 0.03	0.478 ± 0.03
GPT-4 [31]	0	0.257 ± 0.04	0.404 ± 0.06	0.313 ± 0.05	0.342 ± 0.06	0.537 ± 0.07	0.417 ± 0.06	0.410 ± 0.05	0.644 ± 0.06	0.500 ± 0.05
	1	0.272 ± 0.05	0.437 ± 0.07	0.335 ± 0.06	0.354 ± 0.07	0.569 ± 0.09	0.436 ± 0.08	0.417 ± 0.06	0.671 ± 0.06	0.514 ± 0.06
	5	0.327 ± 0.06	0.460 ± 0.06	0.382 ± 0.06	0.414 ± 0.07	0.583 ± 0.07	0.484 ± 0.07	0.480 ± 0.06	0.670 ± 0.05	0.561 ± 0.06
Llama-2-7B Chat [11]	0	0.157 ± 0.03	0.295 ± 0.05	0.205 ± 0.03	0.211 ± 0.03	0.396 ± 0.06	0.275 ± 0.04	0.255 ± 0.03	0.479 ± 0.05	0.332 ± 0.03
	1	0.172 ± 0.03	0.317 ± 0.05	0.223 ± 0.04	0.221 ± 0.03	0.409 ± 0.06	0.287 ± 0.04	0.269 ± 0.03	0.497 ± 0.05	0.349 ± 0.03
	5	0.197 ± 0.03	0.334 ± 0.05	0.247 ± 0.04	0.264 ± 0.04	0.449 ± 0.06	0.332 ± 0.05	0.312 ± 0.03	0.530 ± 0.05	0.392 ± 0.03
Llama-2-13B Chat	0	0.177 ± 0.04	0.265 ± 0.06	0.212 ± 0.05	0.231 ± 0.06	0.345 ± 0.07	0.277 ± 0.06	0.280 ± 0.05	0.420 ± 0.07	0.336 ± 0.06
	1	0.158 ± 0.02	0.310 ± 0.04	0.209 ± 0.03	0.220 ± 0.03	0.432 ± 0.06	0.291 ± 0.04	0.267 ± 0.02	0.525 ± 0.03	0.354 ± 0.02
	5	0.186 ± 0.02	0.300 ± 0.03	0.229 ± 0.02	0.260 ± 0.03	0.419 ± 0.04	0.321 ± 0.03	0.317 ± 0.02	0.511 ± 0.03	0.391 ± 0.02
Llama-2-70B Chat	0	0.218 ± 0.05	0.192 ± 0.03	0.202 ± 0.03	0.286 ± 0.06	0.254 ± 0.04	0.267 ± 0.04	0.337 ± 0.05	0.300 ± 0.04	0.314 ± 0.04
	1	0.171 ± 0.04	0.329 ± 0.07	0.225 ± 0.05	0.231 ± 0.04	0.443 ± 0.08	0.303 ± 0.05	0.278 ± 0.03	0.535 ± 0.05	0.366 ± 0.04
	5	0.200 ± 0.03	0.383 ± 0.05	0.263 ± 0.04	0.268 ± 0.04	0.513 ± 0.07	0.352 ± 0.05	0.320 ± 0.03	0.614 ± 0.05	0.420 ± 0.03

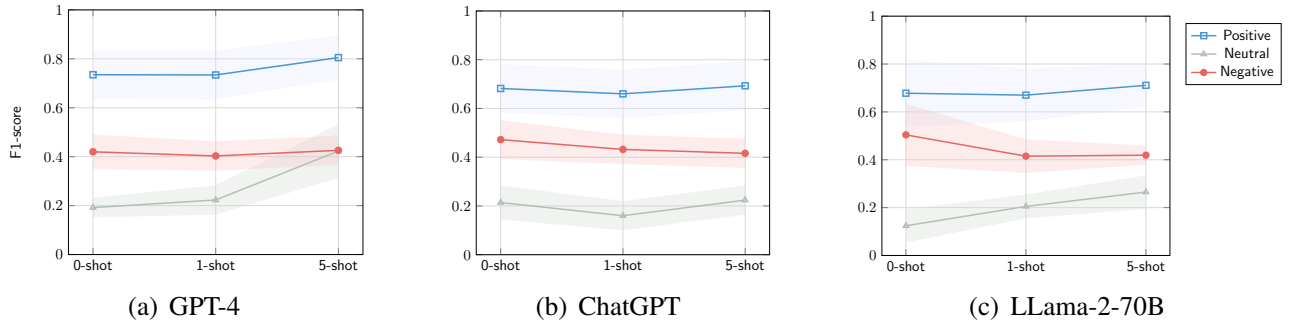


Fig. 3: Comparison of *zero-shot*, *1-shot*, and *5-shot* performances of GPT4, ChatGPT, and LLama-70B in predicting feature-specific sentiment. *5-shot* shows an increase of 7% and 3% in the f1-score of GPT-4 and LLama-70B for predicting *positive* sentiment. For the *neutral* sentiment, GPT-4 and LLama-70B f1 performance is improved by 23% and 14% with *5-shot*.

in ChatGPT. The highest gain of 8.5% average f1-score over its *zero-shot* performance is achieved by LLama-70B. Among all LLM models, GPT-4 is the best model with an f1-score of 48.4%, indicating a 24% improvement over the SAFE approach with *5-shot*. Again, the fine-tuned RE-BERT model performs 7% better than GPT-4 in terms of f1-score.

Relaxing the matching strategy to 2-word and *1-shot* improved the precision and recall of LLM models, with the exception of ChatGPT. The average f1 performance of ChatGPT decreased by 1%, but *5-shot* has improved it by 3.5%. Compared to its *zero-shot* performance, with *1-shot*, the LLama-70B model achieved the highest gain of 8.5% in the f1-score, and *5-shot* has further boosted it to 10.6%. Similarly, *1-shot* improved the f1 performance of LLama-7B and LLama-13B by 4%, and *5-shot* further increased it to 6%. GPT-4 performance has increased 1.4% in f1-score with *1-shot*, while the *5-shot* has increased it by 6%. GPT-4 with *5-shot* has resulted in the f1-score of 56.1%, which is an improvement of

23% over the SAFE approach, but the fine-tuned RE-BERT model is still better than GPT-4 by 6% f1-score.

Finding 3: With 5-shot learning, GPT-4 improved the f1-score by 6% (i.e., 56%), representing a 23% improvement over the SAFE approach. However, the fine-tuned RE-BERT still outperforms GPT-4 by 6% in f1-score.

5.2.2 Feature-specific sentiment prediction performance: As previously described in section 5.1.2, we only considered the set of true positive features obtained in RQ2 and constructed three datasets, each corresponding to true positive features and corresponding sentiments from the exact match, partial match ($n = 1$), and partial match ($n = 2$).

In line with the reasoning provided in section 5.1.2 and constrained by limited space, Figure 3 compares the results of feature-specific sentiment prediction for individual sentiment polarity using the partial match ($n = 2$) only for larger models

such as GPT-4, ChatGPT, and LLama-70B.

It is evident from Figure 3, for predicting the *positive* sentiment, compared to *zero-shot*, the average f1 performances of LLMs is decreased by up to 2%. GPT-4 f1 performance almost remains the same. *5-shot* has slightly improved the f1 performance of ChatGPT (*i.e.* 1%) but GPT-4 and LLama-70B have shown an increase of 7% and 3%, respectively. Under *5-shot* settings, GPT-4 has achieved the best f1-score of 80.5% for predicting *positive* sentiment.

For the *neutral* sentiment (see Figure 3), ChatGPT performance dropped by 5% with *1-shot*, but GPT-4 and LLama-70B yielded a gain of 3% and 8% in average f1-score, respectively. Compared to *zero-shot* performance, *5-shot* has improved the performance of all LLM models. Interestingly, the average f1-score of GPT-4 and LLama-70B is increased by 23% and 14%, respectively, over its *zero-shot* performance. Compared to *zero-shot* scenario, *5-shot* improved the performances ChatGPT by 1%. Under *5-shot* settings, GPT-4 has obtained the best f1-score of 42% for predicting *neutral* sentiment.

As it can be seen in Figure 3, for the *negative* sentiment, implementing *5-shot*, GPT-4’s performance remained the same as *zero-shot*, whereas *1-shot* decreased its performance by 1.7%. The f1-performance for ChatGPT decreased by 5.6% with *1-shot* and further declined by 1.6% when *5-shot* was used. With *1-shot*, the highest drop of 8% in average f1 performance is shown by LLama-70B, and with *5-shot*, the f1 performance almost remains the same as the *1-shot*. For the prediction of *negative* sentiment, the best f1-score of almost 44% is attained by LLama-70B with *0-shot*.

Finding 4: For neutral sentiment prediction, *5-shot* improves the f1-score by 23% for GPT-4 and 14% for LLama-2-70B. In positive sentiment prediction, *5-shot* improves the f1-score by 7% for GPT-4 and 3% for LLama-70b. However, for negative sentiment, *5-shot* does not improve the performance of GPT-4, and f1-score decreases for ChatGPT and LLama-2-70B by 7% and 8%, respectively.

6 DISCUSSION

6.1 Implications for Research

First, this section discusses the results of open-source and proprietary LLM models in extracting app features from user reviews. Next, we present an error analysis of LLMs using a sample of reviews. Finally, we examine the generalization capabilities of LLMs by comparing performance of LLMs against each individual app in the evaluation dataset.

Open-source versus Proprietary LLMs: Our findings show that the open-source LLama-70B model stands as the most comparable counterpart to proprietary GPT family models. Looking at Table 3 in Section 5, it is evident that, compared to the proprietary LLM models ChatGPT and GPT-4, the open-source LLama-2-70B shows the most substantial increase in f1-score with *few-shot* learning. This underscores the potential

of open-source models to approach the performance levels of closed-source ChatGPT and GPT-4 through in-context learning or knowledge distillation techniques. In our *few-shot* experiments, we employed a random selection of examples for in-context learning; however, we hypothesize that selecting semantically similar examples to the input review might further enhance the performance of LLama-70B models. In addition to providing detailed instructions, our L-prompt employed to answer RQ1 includes examples of functional features. Interestingly, we notice that only the *zero-shot* performance of LLama-70B with L-prompt surpasses the *1-shot* performance of other LLM models by a margin of 3% in terms of f1-score.

Error Analysis of LLMs: As the results clearly showed, LLMs struggled in accurately extracting app features and sentiment information from user reviews. To understand the factors contributing to inaccurate feature and sentiment predictions, we randomly selected 25 reviews and examined the predictions of LLama-70B, ChatGPT, and GPT-4 models using the partial match 2 evaluation strategy. In Table 4, we showcase the predictions of these models on six review sentences (R1 to R6), showing the common issues encountered in feature extraction and sentiment prediction. Within each review, features labeled by humans are highlighted and enclosed in brackets, and true sentiment polarities are indicated as POS (for *positive*), NEU (for *neutral*), and NEG (for *negative*). The table shows the predictions of features or sentiments by the LLM models, with a symbol ✓ indicating a correct prediction and a symbol ✗ indicating an incorrect one. It is evident that LLMs encountered difficulties in understanding what constitutes an app feature, as they incorrectly identified words such as *bugs*, *force crashes*, and *app* as features in the first and second review examples (*i.e.* R1 and R2). In the third review example R3, the phrase *easy to use* is extracted as a potential feature, which is used to praise the non-functional aspect of the app. In the context of app feature extraction, the feature "add filtets w/ my pics" in the fourth review example R4 was annotated by humans with the words "filtets" and "pics" (misspelled), and "w/" (an abbreviation for "with"). Interestingly, LLama-70B also extracted the same feature as human-annotated but corrected the spelling mistakes and expanded the abbreviation. Both ChatGPT and GPT-4 also extracted the same feature but corrected the spelling of "filtets" to "filters". The ability of LLMs to perform multiple tasks, such as correcting spelling and abbreviations in user reviews, requires either the creation of new evaluation benchmarks or the adoption of alternative evaluation methods distinct from traditional approaches

In the case of review example R4, it seems GPT models preferred extracting shorter features such as "add filters", but in the review example R5, the same GPT models opted to predict longer features, such as "login & log out feature," that span multiple human-labeled features. Looking at LLM’s predictions against example user review R6, LLama-70B identified the feature "tempo/speed control" and ChatGPT detected the feature "add temp/speed control" but the word "control" does

TABLE 4: Feature-sentiment pairs extracted by LLama-2-70B, ChatGPT, and GPT-4 models from user reviews (R1 to R6). Human-labeled features in reviews are highlighted and enclosed in brackets. POS, NEU, and NEG show true sentiment polarity labels in reviews. The ✗ symbol indicates an incorrect prediction, while the ✓ symbol indicates a correct prediction using the partial match 2 evaluation strategy.

Review	LLama-2-70B Chat		ChatGPT		GPT-4	
	App feature	Sentiment	App feature	Sentiment	App feature	Sentiment
R1 -> So many bugs.force crashes and [messages cannot be sent] _{NEU}	bugs ✗ force crashes ✗ messages cannot be sent ✓	NEG ✗ NEG ✗ NEG ✓	bugs ✗ force crashes ✗ messages cannot be sent ✓	NEG ✗ NEG ✗ NEG ✓	bugs ✗ force crashes ✗ messages cannot be sent ✓	NEG ✗ NEG ✗ NEG ✓
R2 -> Best app in world	app ✗	POS ✗	best app ✗	POS ✗	None ✓	None ✓
R3 -> Its easy to use and has a good [user interface] _{POS}	easy to use ✗ good user interface ✓	POS ✗ POS ✓	easy to use ✗ good user interface ✓	POS ✗ POS ✓	easy to use ✗ user interface ✓	POS ✗ POS ✓
R4 -> I cant [add fillets w/ my pics] _{NEU} with the lates version using my galaxy pls fix this!	add filters with pictures ✗	NEG ✗	add filters ✗	NEG ✗	add filters ✗	NEG ✗
R5 -> Need to be [login] _{NEU} & [log out] _{NEU} feature for security reason or [password] _{NEU} option	login & log out feature ✗ password option ✓	NEU ✓ NEU ✓	login & log out feature ✗ password option ✓	POS ✗ POS ✗	login & log out feature ✗ password option ✓	POS ✗ POS ✗
R6 -> I love it I think they should add is a tempo/speed thing so you can [listen at different speeds] _{NEU} that would be cool.	tempo/speed control ✗	POS ✗	add tempo/speed control ✗ listen at different speeds ✓	POS ✗ POS ✗	tempt/speed thing ✗	POS ✗

not exist in the review text, and both models confabulated it themselves. It is important to note that ChatGPT extracted non-consecutive feature words like "add temp/speed control" by omitting the functioning verb "is" and the article "a" from the extracted feature. Since users can express app features as non-consecutive words, this capability of LLMs is beneficial over sequence prediction models such as CRF and LSTM that can only extract consecutive words as features [32].

In the context of feature-specific sentiment prediction, we have observed that LLM models often confuse the sentiment *neutral* with *negative*. For instance, for the first review R1 and the fourth review R4, labeled sentiment for features such as "messages cannot be sent" and "add filters w/ my pics" is predicted as *negative*, when in reality it is *neutral*. Upon investigation, it was discovered that the annotation guidelines used for review annotation instructed annotators to assign *negative* or *positive* sentiment only when a clear expression of *positive* or *negative* opinion is present towards a feature. As per these instructions, app users in the first review R1 and fourth review R4 stated that they could not utilize a functionality without explicitly conveying their sentiments towards these features. In the fifth example review R5, Llama-70B model correctly predicted the *neutral* sentiment, whereas the GPT family wrongly predicted it as *positive*.

Generalization capabilities of LLMs: To assess the capabilities of LLMs in conducting feature-specific sentiment analysis across different applications, we performed a comparison of the 5-shot f1 performance scores¹³ of three LLM models: GPT-4, ChatGPT, and LLama-2-70B across eight different applications using a spider chart shown in Figure 4. This analysis reveals that LLM models exhibit varying performance across different applications. Specifically, LLM models demonstrate improved performance in extracting features from user reviews of applications such as "WhatsApp," "Twitter," and "PhotoEditor". Notably, all models display a low f1-score for the "Netflix" application. This observed trend may

¹³The f1-score for each model is calculated as the average across three feature matching strategies

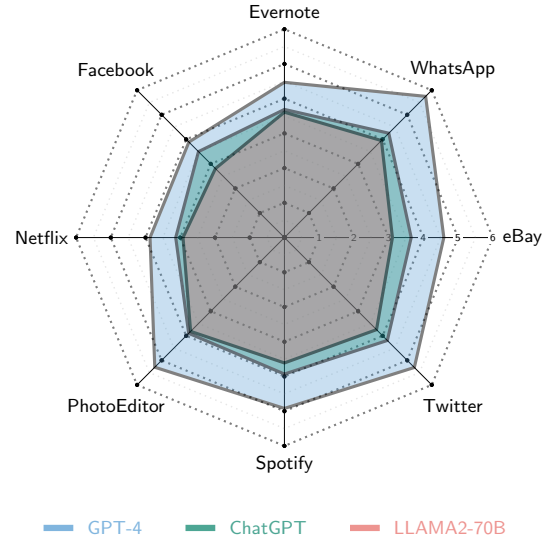


Fig. 4: Comparison of 5-shot f1 performance of GPT-4, ChatGPT, LLama-2-70B Chat for individual app.

be associated with the proportion of review data from each application employed in training or fine-tuning these models.

6.2 Implications for Practitioners

Our findings indicate that while LLMs show promise, their precision and recall in extracting app features from user reviews are not yet adequate for practical implementation. Although prompt engineering demonstrates lower performance compared to fine-tuned pretrained models, it offers a more cost-efficient approach for practitioners, particularly in settings where labeled data is scarce. Using LLMs for prompting is highly suited to extracting feature-sentiment data from a range of feedback sources, including in-app feedback, community forums, emails, and Reddit, which all demand a broad understanding of language.

7 THREATS TO VALIDITY

This section discusses key threats to external validity of this study by focusing on prompt engineering, the non-deterministic behavior of large language models (LLMs), the evolving nature of LLMs, the influence of parameters on LLM performance, and the labeled review dataset.

Prompt Engineering: Our study used prompts to extract feature-sentiment pairs from user reviews. We understand that the quality of the prompts used can significantly influence the results of our study. To address this potential concern, we followed the best practices of prompt design as recommended in the existing literature. Furthermore, we rigorously evaluated the effectiveness of our prompts by subjecting them to testing on sample reviews. Additionally, in addressing RQ1, we utilized two types of prompts - short and long - and performed a comparative analysis of their effectiveness in extracting app features and feature-specific sentiments. Despite presenting our findings, we acknowledge that one of the limitations of prompting LLMs is that they may struggle to understand the full context and intricacies of a given prompt, especially when it comes to highly specific or specialized information.

Non-determinism in LLMs: It is well known that LLMs hallucinate [33], and their output is non-deterministic. To mitigate this concern, we performed three iterations of each LLM on the entire review dataset to quantify the uncertainty inherent in the models' responses. We then reported the average score along with the standard deviation for each performance metric. **LLMs Evolutionary Landscape:** Our study assessed both open-source and closed-source state-of-the-art Large Language Models (LLMs). However, the landscape of LLMs is rapidly evolving, making it impossible to evaluate all models in a single study, considering the substantial computational and financial resources needed. Therefore, it is yet to be investigated whether the results generalize to other LLMs.

Impact of Temperature and Token Limits on LLM Performance: In our study, all models were evaluated with the parameters temperature set to 0 and max_new_tokens set to 1000. As a result, further investigation is required to understand the impact of varying these parameters on the reported performance of the models for this task.

Labeled Review Dataset: Our evaluation study relies on a review dataset comprising 1000 labeled reviews from 8 distinct applications. We acknowledge the relatively small size of this dataset. However, to the best of our knowledge, it is the only openly available review dataset with annotated feature and sentiment pairs.

8 CONCLUSIONS AND FUTURE WORK

Automated analysis of users' opinions regarding features in user feedback can help understand users' perceptions of functionality. Our study has evaluated the *zero-shot* and *few-shot* performance of LLMs such as GPT-4, ChatGPT, and Llama-2-Chat variants, in extracting app features and associated sentiments from app reviews simultaneously. We conducted a comparative analysis of LLMs' performances against rule-based and supervised learning methods.

In *zero-shot* learning, GPT-4 is the top-performing model for extracting app features, surpassing rule-based approaches by 23.6% in f1-score with the exact matching strategy. However, the fine-tuned BERT model still outperforms GPT-4 by 14% in the f1-score. In the *5-shot* scenario, both GPT-4 and Llama-2-70B show further improvements of 7% and 6% in f1-score, respectively. Our evaluation study has showcased the encouraging performance of LLMs in extracting fine-grained information from user feedback. This capability of LLMs holds promise in assisting developers with their software maintenance and evolution activities by analyzing users' opinions.

In our future work, we aim to investigate the performance of small-scale models trained to replicate the behavior of larger, and complex models using knowledge distillation techniques [34]. Additionally, a Chain-of-Thought [35] (CoT) prompting strategy could enhance performance in extracting feature-sentiment information as it organizes input in a manner that mimics human reasoning. Finally, exploring parameter-efficient fine-tuning of small LLMs through the Low-Rank Adaptation [36] (*i.e.* LoRA) technique is another promising direction for achieving human-level performance on this task.

ACKNOWLEDGMENT

This work has received funding from the EU H2020 program under the SoBigData++ project (grant agreement No. 871042), CHIST-ERA grant No. CHIST-ERA-19-XAI-010, grant PRG1226 of the Estonian Research Council, and the HAMISON project.

REFERENCES

- [1] B. Lin, N. Cassee, A. Serebrenik, G. Bavota, N. Novielli, and M. Lanza, "Opinion Mining for Software Development: A Systematic Literature Review," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 3, 3 2022. [Online]. Available: <https://dl.acm.org/doi/10.1145/3490388>
- [2] J. Dąbrowski, E. Letier, A. Perini, and A. Susi, "Analysing app reviews for software engineering: a systematic literature review," *Empirical Software Engineering*, vol. 27, no. 2, pp. 1–63, 3 2022. [Online]. Available: <https://link.springer.com/article/10.1007/s10664-021-10065-7>
- [3] E. Guzman and W. Maalej, "How do users like this feature? A fine grained sentiment analysis of App reviews," *2014 IEEE 22nd International Requirements Engineering Conference, RE 2014 - Proceedings*, pp. 153–162, 9 2014.
- [4] F. Dalpiaz and M. Parente, "RE-SWOT: From User Feedback to Requirements via Competitor Analysis," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11412 LNCS, pp. 55–70, 2019. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-15538-4_4
- [5] F. A. Shah, Y. Sabanin, and D. Pfahl, "Feature-Based evaluation of competing apps," *WAMA 2016 - Proceedings of the International Workshop on App Market Analytics, co-located with FSE 2016*, pp. 15–21, 11 2016.
- [6] M. Dragoni, M. Federici, and A. Rexha, "An unsupervised aspect extraction strategy for monitoring real-time reviews stream," *Information processing & management*, vol. 56, no. 3, pp. 1103–1118, 2019.
- [7] T. Johann, C. Stanik, A. M. Alizadeh, and W. Maalej, "SAFE: A Simple Approach for Feature Extraction from App Descriptions and App Reviews," *Proceedings - 2017 IEEE 25th International Requirements Engineering Conference, RE 2017*, pp. 21–30, 9 2017.
- [8] A. F. De Araújo and R. M. Marcacini, "RE-BERT: Automatic extraction of software requirements from app reviews using BERT language model," *Proceedings of the ACM Symposium on Applied Computing*, vol. 7, no. 21, pp. 1321–1327, 3 2021.

- [9] Q. Motger, A. Miaschi, F. Dell’Orletta, X. Franch, and J. Marco, “T-frex: A transformer-based feature extraction method from mobile app reviews,” *arXiv preprint arXiv:2401.03833*, 2024.
- [10] OpenAI, “Chatgpt: Optimizing language models for dialogue,” 2022. [Online]. Available: <https://openai.com/blog/chatgpt>
- [11] Meta, “Open source models from meta,” 2023. [Online]. Available: <https://llama.meta.com/>
- [12] Y. Liu, T. Han, S. Ma, J. Zhang, Y. Yang, J. Tian, H. He, A. Li, M. He, Z. Liu *et al.*, “Summary of chatgpt-related research and perspective towards the future of large language models,” *Meta-Radiology*, p. 100017, 2023.
- [13] J. Zhang, Y. Chen, N. Niu, Y. Wang, and C. Liu, “Empirical Evaluation of ChatGPT on Requirements Information Retrieval Under Zero-Shot Setting,” 4 2023. [Online]. Available: <https://arxiv.org/abs/2304.12562v2>
- [14] N. Chen, J. Lin, S. C. Hoi, X. Xiao, and B. Zhang, “Ar-miner: mining informative reviews for developers from mobile app marketplace,” in *Proceedings of the 36th international conference on software engineering*, 2014, pp. 767–778.
- [15] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, “Language models are few-shot learners,” *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.
- [16] S. Min, X. Lyu, A. Holtzman, M. Artetxe, M. Lewis, H. Hajishirzi, and L. Zettlemoyer, “Rethinking the role of demonstrations: What makes in-context learning work?” *arXiv preprint arXiv:2202.12837*, 2022.
- [17] X. Gu and S. Kim, “What parts of your apps are loved by users?” *Proceedings - 2015 30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015*, pp. 760–770, 1 2016.
- [18] F. A. Shah, K. Sirts, and D. Pfahl, “Using app reviews for competitive analysis: Tool support,” *WAMA 2019 - Proceedings of the 3rd ACM SIGSOFT International Workshop on App Market Analytics, co-located with ESEC/FSE 2019*, pp. 40–46, 8 2019.
- [19] M. K. Uddin, Q. He, J. Han, and C. Chua, “Mining Cross-Domain Apps for Software Evolution: A Feature-based Approach,” *Proceedings - 2021 36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021*, pp. 743–755, 2021.
- [20] M. Assi, S. Hassan, Y. Tian, and Y. Zou, “FeatCompare: Feature comparison for competing mobile apps leveraging user reviews,” *Empirical Software Engineering*, vol. 26, no. 5, pp. 1–38, 9 2021.
- [21] P. M. Vu, T. T. Nguyen, H. V. Pham, and T. T. Nguyen, “Mining user opinions in mobile app reviews: A keyword-based approach,” *Proceedings - 2015 30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015*, pp. 749–759, 1 2016.
- [22] M. Tushev, F. Ebrahimi, and A. Mahmoud, “Domain-Specific Analysis of Mobile App Reviews Using Keyword-Assisted Topic Models,” *Proceedings - International Conference on Software Engineering*, vol. 2022-May, pp. 762–773, 2022.
- [23] H. Guo and M. P. Singh, “Caspar: Extracting and synthesizing user stories of problems from app reviews,” *Proceedings - International Conference on Software Engineering*, pp. 628–640, 6 2020.
- [24] N. H. Bakar, Z. M. Kasirun, N. Salleh, and H. A. Jalab, “Extracting features from online software reviews to aid requirements reuse,” *Applied Soft Computing*, vol. 49, pp. 1297–1315, 12 2016.
- [25] J. Dąbrowski, E. Letier, A. Perini, and A. Susi, “Mining and searching app reviews for requirements engineering: Evaluation and replication studies,” *Information Systems*, vol. 114, p. 102181, 3 2023.
- [26] F. A. Shah, K. Sirts, and D. Pfahl, “Is the SAFE Approach Too Simple for App Feature Extraction? A Replication Study,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11412 LNCS, pp. 21–36, 2019. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-15538-4_2
- [27] Y. Wang, J. Wang, H. Zhang, X. Ming, L. Shi, and Q. Wang, “Where is Your App Frustrating Users?” *Proceedings - International Conference on Software Engineering*, vol. 2022-May, pp. 2427–2439, 2022.
- [28] V. Krishnan and V. Ganapathy, “Named entity recognition,” *Stanford Lecture CS229*, 2005.
- [29] H. Wu, W. Deng, X. Niu, and C. Nie, “Identifying key features from app user reviews,” *Proceedings - International Conference on Software Engineering*, pp. 922–932, 5 2021.
- [30] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, “Training language models to follow instructions with human feedback,” *Advances in neural information processing systems*, vol. 35, pp. 27 730–27 744, 2022.
- [31] OpenAI, “Gpt-4 technical report,” 2023. [Online]. Available: <https://arxiv.org/abs/2303.08774>
- [32] F. A. Shah, K. Sirts, and D. Pfahl, “The Impact of Annotation Guidelines and Annotated Data on Extracting App Features from App Reviews,” 10 2018. [Online]. Available: <http://arxiv.org/abs/1810.05187>
- [33] Y. Bang, S. Cahyawijaya, N. Lee, W. Dai, D. Su, B. Wilie, H. Lovenia, Z. Ji, T. Yu, W. Chung *et al.*, “A multitask, multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity,” *arXiv preprint arXiv:2302.04023*, 2023.
- [34] Y. Gu, L. Dong, F. Wei, and M. Huang, “Minillm: Knowledge distillation of large language models,” in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: <https://openreview.net/forum?id=5h0qf7IBZZ>
- [35] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, “Chain-of-thought prompting elicits reasoning in large language models,” *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.
- [36] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, “Lora: Low-rank adaptation of large language models,” *arXiv preprint arXiv:2106.09685*, 2021.

APPENDIX A

DETAILED RESULTS OF SENTIMENT PREDICTION PERFORMANCE OF LLMs WITH VARYING LEVELS OF FEATURE-EXTRACTION MATCHING STRATEGIES

TABLE 5: Comparison of sentiment prediction performances of LLMs with S-prompt and L-prompt and exact and partial feature-matching strategies.(Second best result and Third best result). Note that SAFE and RE-BERT techniques cannot predict sentiment towards app features (see Section 4.3).

Model	Prompt	Feature matching	Positive sentiment			Neutral sentiment			Negative sentiment		
			Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
GuMa [3]		Exact (n = 0)	0.68	0.84	0.75	0.68	0.40	0.50	0.41	0.64	0.50
		Exact (n = 1)	0.61	0.89	0.75	0.85	0.49	0.62	0.40	0.68	0.50
		Exact (n = 2)	0.59	0.89	0.71	0.86	0.54	0.66	0.38	0.61	0.47
ReUS [6]		Exact (n = 0)	0.90	0.62	0.73	0.60	0.87	0.71	0.62	0.36	0.46
		Exact (n = 1)	0.80	0.48	0.60	0.66	0.88	0.75	0.43	0.24	0.31
		Exact (n = 2)	0.80	0.44	0.57	0.68	0.91	0.78	0.41	0.18	0.25
ChatGPT [10]	S	Exact (n = 0)	0.515 ± 0.02	0.986 ± 0.02	0.663 ± 0.01	0.859 ± 0.21	0.133 ± 0.06	0.227 ± 0.10	0.303 ± 0.09	0.945 ± 0.08	0.452 ± 0.10
		Partial (n = 1)	0.528 ± 0.11	0.981 ± 0.04	0.679 ± 0.09	0.956 ± 0.06	0.125 ± 0.04	0.219 ± 0.07	0.335 ± 0.08	0.958 ± 0.06	0.492 ± 0.09
		Partial (n = 2)	0.531 ± 0.12	0.987 ± 0.02	0.682 ± 0.10	0.950 ± 0.07	0.122 ± 0.04	0.214 ± 0.07	0.317 ± 0.07	0.957 ± 0.06	0.472 ± 0.08
ChatGPT	L	Exact (n = 0)	0.472 ± 0.14	0.994 ± 0.02	0.627 ± 0.11	0.917 ± 0.13	0.151 ± 0.07	0.252 ± 0.11	0.295 ± 0.08	0.950 ± 0.07	0.444 ± 0.10
		Partial (n = 1)	0.481 ± 0.12	0.989 ± 0.02	0.639 ± 0.09	0.921 ± 0.10	0.140 ± 0.05	0.240 ± 0.09	0.312 ± 0.07	0.958 ± 0.06	0.466 ± 0.08
		Partial (n = 2)	0.484 ± 0.13	0.991 ± 0.02	0.638 ± 0.12	0.915 ± 0.11	0.138 ± 0.04	0.238 ± 0.06	0.302 ± 0.05	0.951 ± 0.06	0.456 ± 0.06
GPT-4 [31]	S	Exact (n = 0)	0.579 ± 0.15	0.973 ± 0.03	0.714 ± 0.12	0.852 ± 0.17	0.121 ± 0.04	0.210 ± 0.06	0.289 ± 0.10	0.974 ± 0.03	0.434 ± 0.12
		Partial (n = 1)	0.596 ± 0.13	0.980 ± 0.03	0.733 ± 0.10	0.871 ± 0.16	0.110 ± 0.03	0.195 ± 0.05	0.281 ± 0.06	0.978 ± 0.03	0.432 ± 0.07
		Partial (n = 2)	0.597 ± 0.12	0.985 ± 0.02	0.735 ± 0.10	0.895 ± 0.12	0.108 ± 0.03	0.192 ± 0.04	0.270 ± 0.06	0.981 ± 0.02	0.420 ± 0.07
GPT-4	L	Exact (n = 0)	0.585 ± 0.12	0.963 ± 0.05	0.719 ± 0.10	0.920 ± 0.08	0.265 ± 0.05	0.408 ± 0.06	0.256 ± 0.07	0.944 ± 0.06	0.397 ± 0.08
		Partial (n = 1)	0.625 ± 0.11	0.972 ± 0.04	0.754 ± 0.08	0.928 ± 0.07	0.245 ± 0.05	0.385 ± 0.06	0.273 ± 0.06	0.956 ± 0.04	0.421 ± 0.07
		Partial (n = 2)	0.629 ± 0.10	0.979 ± 0.02	0.761 ± 0.07	0.937 ± 0.06	0.241 ± 0.04	0.381 ± 0.05	0.266 ± 0.05	0.953 ± 0.05	0.413 ± 0.06
Llama-2-7B Chat [11]	S	Exact (n = 0)	0.549 ± 0.20	0.964 ± 0.04	0.679 ± 0.17	0.797 ± 0.15	0.199 ± 0.08	0.311 ± 0.10	0.295 ± 0.15	0.770 ± 0.23	0.440 ± 0.16
		Partial (n = 1)	0.553 ± 0.15	0.976 ± 0.02	0.693 ± 0.13	0.787 ± 0.18	0.187 ± 0.07	0.297 ± 0.10	0.314 ± 0.10	0.866 ± 0.13	0.446 ± 0.10
		Partial (n = 2)	0.532 ± 0.15	0.974 ± 0.02	0.674 ± 0.14	0.832 ± 0.08	0.192 ± 0.06	0.306 ± 0.09	0.309 ± 0.07	0.850 ± 0.12	0.444 ± 0.07
Llama-2-7B Chat	L	Exact (n = 0)	0.481 ± 0.16	0.980 ± 0.03	0.627 ± 0.16	0.858 ± 0.13	0.161 ± 0.06	0.263 ± 0.08	0.323 ± 0.10	0.781 ± 0.25	0.415 ± 0.10
		Partial (n = 1)	0.490 ± 0.13	0.984 ± 0.02	0.643 ± 0.13	0.860 ± 0.10	0.151 ± 0.06	0.250 ± 0.09	0.310 ± 0.11	0.798 ± 0.21	0.408 ± 0.08
		Partial (n = 2)	0.485 ± 0.13	0.980 ± 0.02	0.637 ± 0.12	0.870 ± 0.09	0.160 ± 0.03	0.266 ± 0.05	0.306 ± 0.11	0.801 ± 0.20	0.413 ± 0.11
Llama-2-13B Chat	S	Exact (n = 0)	0.532 ± 0.17	0.968 ± 0.03	0.668 ± 0.15	0.432 ± 0.33	0.043 ± 0.05	0.077 ± 0.09	0.272 ± 0.12	0.994 ± 0.02	0.413 ± 0.14
		Partial (n = 1)	0.562 ± 0.14	0.975 ± 0.02	0.701 ± 0.13	0.497 ± 0.27	0.047 ± 0.04	0.085 ± 0.07	0.279 ± 0.09	0.994 ± 0.01	0.428 ± 0.10
		Partial (n = 2)	0.552 ± 0.13	0.980 ± 0.01	0.696 ± 0.12	0.598 ± 0.28	0.051 ± 0.05	0.092 ± 0.08	0.273 ± 0.06	0.990 ± 0.02	0.423 ± 0.08
Llama-2-13B Chat	L	Exact (n = 0)	0.496 ± 0.16	0.973 ± 0.03	0.640 ± 0.15	0.870 ± 0.11	0.163 ± 0.09	0.263 ± 0.12	0.340 ± 0.10	0.947 ± 0.05	0.487 ± 0.12
		Partial (n = 1)	0.529 ± 0.15	0.976 ± 0.03	0.672 ± 0.13	0.883 ± 0.10	0.148 ± 0.08	0.243 ± 0.11	0.286 ± 0.05	0.950 ± 0.05	0.436 ± 0.06
		Partial (n = 2)	0.533 ± 0.15	0.981 ± 0.02	0.676 ± 0.13	0.897 ± 0.09	0.156 ± 0.07	0.257 ± 0.10	0.297 ± 0.04	0.949 ± 0.05	0.450 ± 0.04
Llama-2-70B Chat	S	Exact (n = 0)	0.513 ± 0.21	0.955 ± 0.11	0.647 ± 0.21	0.722 ± 0.27	0.074 ± 0.04	0.131 ± 0.07	0.313 ± 0.22	0.880 ± 0.22	0.428 ± 0.23
		Partial (n = 1)	0.541 ± 0.17	0.997 ± 0.01	0.684 ± 0.15	0.722 ± 0.27	0.062 ± 0.04	0.113 ± 0.07	0.341 ± 0.14	0.968 ± 0.07	0.485 ± 0.15
		Partial (n = 2)	0.530 ± 0.16	0.998 ± 0.01	0.678 ± 0.14	0.727 ± 0.28	0.069 ± 0.04	0.124 ± 0.07	0.355 ± 0.13	0.967 ± 0.08	0.504 ± 0.13
Llama-2-70B Chat	L	Exact (n = 0)	0.507 ± 0.17	0.992 ± 0.02	0.652 ± 0.16	0.984 ± 0.02	0.199 ± 0.07	0.323 ± 0.09	0.326 ± 0.15	0.969 ± 0.04	0.465 ± 0.15
		Partial (n = 1)	0.522 ± 0.17	0.983 ± 0.04	0.662 ± 0.16	0.972 ± 0.04	0.197 ± 0.04	0.325 ± 0.05	0.292 ± 0.07	0.978 ± 0.03	0.444 ± 0.08
		Partial (n = 2)	0.519 ± 0.15	0.989 ± 0.02	0.664 ± 0.15	0.976 ± 0.03	0.194 ± 0.04	0.320 ± 0.06	0.292 ± 0.06	0.980 ± 0.02	0.446 ± 0.06

Prompt	Type
As an expert information extractor, identify all app features with their corresponding sentiment polarities (i.e., positive, negative or neutral) in the following review text (enclosed in double quotations). Output the results in the format of [('app feature', 'sentiment polarity'), ...]. If no app feature is identified, return an empty Python list. Don't output any other information.	Short
Consider the following definitions of "feature", "feature expression", and "sentiment polarity": The "feature" refers to a software application functionality (e.g., "send message"), a module (e.g., "user account") providing functionalities (e.g., "delete account" or "edit information") or a design component (e.g., UI) providing functional capabilities (e.g., "configuration screen", "button"). The "feature expression" is an actual sequence of words that appears in a review text and explicitly indicate a feature. The "sentiment polarity" refers to the degree of positivity, negativity or neutrality expressed towards the feature of a software application, and the available polarities includes: 'positive', 'neutral' or 'negative'. As an expert information extractor, identify all feature expressions with their corresponding sentiment polarities in the following review text (enclosed in double quotations). Output the results in the format of [('feature expression', 'sentiment polarity'), ...]. If no feature expression is identified, return the empty Python list. Don't output any other information.	Long

TABLE 6: Prompt type

TABLE 7: Comparison of *zero-shot*, *1-shot*, *5-shot* performances of LLMs for predicting feature-specific sentiments using exact and partial feature matching strategies. (Second best result and Third best result). Note that SAFE and RE-BERT techniques cannot predict sentiment towards app features (see Section 4.3).

Model	Shot	Feature matching	Positive sentiment			Neutral sentiment			Negative sentiment		
			Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
GuMa [3]	-	Exact (n = 0)	0.68	0.84	0.75	0.68	0.40	0.50	0.41	0.64	0.50
		Exact (n = 1)	0.61	0.89	0.75	0.85	0.49	0.62	0.40	0.68	0.50
		Exact (n = 2)	0.59	0.89	0.71	0.86	0.54	0.66	0.38	0.61	0.47
ReUS [6]	-	Exact (n = 0)	0.90	0.62	0.73	0.60	0.87	0.71	0.62	0.36	0.46
		Exact (n = 1)	0.80	0.48	0.60	0.66	0.88	0.75	0.43	0.24	0.31
		Exact (n = 2)	0.80	0.44	0.57	0.68	0.91	0.78	0.41	0.18	0.25
ChatGPT [10]	0	Exact (n = 0)	0.515 ± 0.02	0.986 ± 0.02	0.663 ± 0.01	0.859 ± 0.21	0.133 ± 0.06	0.227 ± 0.10	0.303 ± 0.09	0.945 ± 0.08	0.452 ± 0.10
		Partial (n = 1)	0.528 ± 0.11	0.981 ± 0.04	0.679 ± 0.09	0.956 ± 0.06	0.125 ± 0.04	0.219 ± 0.07	0.335 ± 0.08	0.958 ± 0.06	0.492 ± 0.09
		Partial (n = 2)	0.531 ± 0.12	0.987 ± 0.02	0.682 ± 0.10	0.950 ± 0.07	0.122 ± 0.04	0.214 ± 0.07	0.317 ± 0.07	0.957 ± 0.06	0.472 ± 0.08
ChatGPT	1	Exact (n = 0)	0.485 ± 0.15	0.999 ± 0.00	0.637 ± 0.14	0.925 ± 0.11	0.108 ± 0.05	0.189 ± 0.08	0.249 ± 0.04	0.948 ± 0.08	0.391 ± 0.06
		Partial (n = 1)	0.494 ± 0.12	0.999 ± 0.00	0.651 ± 0.10	0.930 ± 0.10	0.094 ± 0.03	0.168 ± 0.06	0.276 ± 0.03	0.959 ± 0.06	0.427 ± 0.04
		Partial (n = 2)	0.501 ± 0.11	0.999 ± 0.00	0.660 ± 0.10	0.935 ± 0.09	0.089 ± 0.03	0.160 ± 0.06	0.280 ± 0.05	0.961 ± 0.06	0.432 ± 0.06
ChatGPT	5	Exact (n = 0)	0.511 ± 0.15	0.996 ± 0.01	0.660 ± 0.13	0.912 ± 0.12	0.144 ± 0.05	0.245 ± 0.08	0.236 ± 0.06	0.944 ± 0.07	0.370 ± 0.08
		Partial (n = 1)	0.537 ± 0.13	0.997 ± 0.01	0.690 ± 0.11	0.940 ± 0.08	0.136 ± 0.04	0.235 ± 0.07	0.268 ± 0.05	0.963 ± 0.05	0.416 ± 0.06
		Partial (n = 2)	0.540 ± 0.12	0.998 ± 0.01	0.693 ± 0.10	0.946 ± 0.07	0.128 ± 0.03	0.224 ± 0.06	0.268 ± 0.05	0.968 ± 0.04	0.416 ± 0.06
GPT-4 [31]	0	Exact (n = 0)	0.579 ± 0.15	0.973 ± 0.03	0.714 ± 0.12	0.852 ± 0.17	0.121 ± 0.04	0.210 ± 0.06	0.289 ± 0.10	0.974 ± 0.03	0.434 ± 0.12
		Partial (n = 1)	0.596 ± 0.13	0.980 ± 0.03	0.733 ± 0.10	0.871 ± 0.16	0.110 ± 0.03	0.195 ± 0.05	0.281 ± 0.06	0.978 ± 0.03	0.432 ± 0.07
		Partial (n = 2)	0.597 ± 0.12	0.985 ± 0.02	0.735 ± 0.10	0.895 ± 0.12	0.108 ± 0.03	0.192 ± 0.04	0.270 ± 0.06	0.981 ± 0.02	0.420 ± 0.07
GPT-4	1	Exact (n = 0)	0.595 ± 0.19	0.981 ± 0.03	0.719 ± 0.17	0.946 ± 0.19	0.153 ± 0.06	0.260 ± 0.08	0.272 ± 0.09	0.985 ± 0.03	0.419 ± 0.10
		Partial (n = 1)	0.599 ± 0.15	0.986 ± 0.02	0.732 ± 0.13	0.940 ± 0.10	0.136 ± 0.05	0.235 ± 0.07	0.268 ± 0.05	0.986 ± 0.02	0.419 ± 0.06
		Partial (n = 2)	0.596 ± 0.13	0.990 ± 0.02	0.734 ± 0.11	0.948 ± 0.09	0.128 ± 0.04	0.223 ± 0.06	0.255 ± 0.05	0.983 ± 0.03	0.403 ± 0.06
GPT-4	5	Exact (n = 0)	0.686 ± 0.14	0.976 ± 0.03	0.797 ± 0.10	0.963 ± 0.05	0.298 ± 0.11	0.444 ± 0.12	0.295 ± 0.08	0.982 ± 0.03	0.446 ± 0.09
		Partial (n = 1)	0.688 ± 0.13	0.979 ± 0.02	0.800 ± 0.10	0.959 ± 0.04	0.275 ± 0.10	0.416 ± 0.12	0.288 ± 0.05	0.986 ± 0.02	0.443 ± 0.06
		Partial (n = 2)	0.691 ± 0.12	0.984 ± 0.02	0.805 ± 0.09	0.960 ± 0.04	0.279 ± 0.10	0.423 ± 0.11	0.274 ± 0.04	0.981 ± 0.03	0.426 ± 0.06
Llama-2-7B Chat [11]	0	Exact (n = 0)	0.549 ± 0.20	0.964 ± 0.04	0.679 ± 0.17	0.797 ± 0.15	0.199 ± 0.08	0.311 ± 0.10	0.295 ± 0.15	0.770 ± 0.23	0.440 ± 0.16
		Partial (n = 1)	0.553 ± 0.15	0.976 ± 0.02	0.693 ± 0.13	0.787 ± 0.18	0.187 ± 0.07	0.297 ± 0.10	0.314 ± 0.10	0.866 ± 0.13	0.446 ± 0.10
		Partial (n = 2)	0.532 ± 0.15	0.974 ± 0.02	0.674 ± 0.14	0.832 ± 0.08	0.192 ± 0.06	0.306 ± 0.09	0.309 ± 0.07	0.850 ± 0.12	0.444 ± 0.07
Llama-2-7B Chat	1	Exact (n = 0)	0.531 ± 0.16	0.942 ± 0.05	0.665 ± 0.14	0.761 ± 0.12	0.177 ± 0.07	0.281 ± 0.09	0.321 ± 0.12	0.820 ± 0.15	0.434 ± 0.10
		Partial (n = 1)	0.552 ± 0.12	0.957 ± 0.03	0.691 ± 0.10	0.775 ± 0.13	0.175 ± 0.07	0.280 ± 0.10	0.314 ± 0.07	0.869 ± 0.14	0.446 ± 0.05
		Partial (n = 2)	0.539 ± 0.11	0.962 ± 0.03	0.684 ± 0.09	0.781 ± 0.11	0.170 ± 0.06	0.274 ± 0.09	0.304 ± 0.06	0.863 ± 0.12	0.439 ± 0.06
Llama-2-7B Chat	5	Exact (n = 0)	0.538 ± 0.17	0.925 ± 0.05	0.659 ± 0.15	0.736 ± 0.19	0.208 ± 0.06	0.321 ± 0.09	0.272 ± 0.11	0.804 ± 0.17	0.389 ± 0.12
		Partial (n = 1)	0.554 ± 0.12	0.919 ± 0.05	0.682 ± 0.09	0.770 ± 0.14	0.206 ± 0.05	0.321 ± 0.08	0.284 ± 0.07	0.864 ± 0.11	0.419 ± 0.07
		Partial (n = 2)	0.547 ± 0.11	0.920 ± 0.05	0.678 ± 0.08	0.751 ± 0.16	0.199 ± 0.06	0.312 ± 0.08	0.277 ± 0.08	0.861 ± 0.10	0.411 ± 0.08
Llama-2-13B Chat	0	Exact (n = 0)	0.532 ± 0.17	0.968 ± 0.03	0.668 ± 0.15	0.432 ± 0.33	0.043 ± 0.05	0.077 ± 0.09	0.272 ± 0.12	0.994 ± 0.02	0.413 ± 0.14
		Partial (n = 1)	0.562 ± 0.14	0.975 ± 0.02	0.701 ± 0.13	0.497 ± 0.27	0.047 ± 0.04	0.085 ± 0.07	0.279 ± 0.09	0.994 ± 0.01	0.428 ± 0.10
		Partial (n = 2)	0.552 ± 0.13	0.980 ± 0.01	0.696 ± 0.12	0.598 ± 0.28	0.051 ± 0.05	0.092 ± 0.08	0.273 ± 0.06	0.990 ± 0.02	0.423 ± 0.08
Llama-2-13B Chat	1	Exact (n = 0)	0.517 ± 0.17	0.946 ± 0.05	0.646 ± 0.16	0.713 ± 0.31	0.082 ± 0.06	0.142 ± 0.10	0.251 ± 0.07	0.952 ± 0.11	0.392 ± 0.10
		Partial (n = 1)	0.527 ± 0.15	0.958 ± 0.03	0.664 ± 0.14	0.686 ± 0.33	0.07 ± 0.05	0.121 ± 0.08	0.244 ± 0.03	0.985 ± 0.03	0.388 ± 0.05
		Partial (n = 2)	0.528 ± 0.15	0.963 ± 0.02	0.668 ± 0.13	0.733 ± 0.34	0.07 ± 0.04	0.124 ± 0.07	0.244 ± 0.03	0.985 ± 0.03	0.390 ± 0.04
Llama-2-13B Chat	5	Exact (n = 0)	0.537 ± 0.19	0.869 ± 0.12	0.648 ± 0.18	0.696 ± 0.33	0.096 ± 0.07	0.162 ± 0.12	0.254 ± 0.09	0.997 ± 0.08	0.394 ± 0.11
		Partial (n = 1)	0.556 ± 0.154	0.925 ± 0.08	0.680 ± 0.14	0.719 ± 0.32	0.084 ± 0.06	0.145 ± 0.10	0.260 ± 0.05	0.998 ± 0.01	0.409 ± 0.06
		Partial (n = 2)	0.559 ± 0.156	0.931 ± 0.08	0.682 ± 0.14	0.716 ± 0.31	0.081 ± 0.06	0.140 ± 0.10	0.259 ± 0.05	0.998 ± 0.06	0.408 ± 0.06
Llama-2-70B Chat	0	Exact (n = 0)	0.513 ± 0.21	0.955 ± 0.11	0.647 ± 0.21	0.722 ± 0.27	0.074 ± 0.04	0.131 ± 0.07	0.313 ± 0.22	0.880 ± 0.22	0.428 ± 0.23
		Partial (n = 1)	0.541 ± 0.17	0.997 ± 0.01	0.684 ± 0.15	0.722 ± 0.27	0.062 ± 0.04	0.113 ± 0.07	0.341 ± 0.14	0.968 ± 0.07	0.485 ± 0.15
		Partial (n = 2)	0.530 ± 0.16	0.998 ± 0.01	0.678 ± 0.14	0.727 ± 0.28	0.069 ± 0.04	0.124 ± 0.07	0.355 ± 0.13	0.967 ± 0.08	0.504 ± 0.13
Llama-2-70B Chat	1	Exact (n = 0)	0.535 ± 0.15	0.939 ± 0.06	0.666 ± 0.13	0.876 ± 0.14	0.115 ± 0.04	0.198 ± 0.06	0.259 ± 0.13	0.931 ± 0.11	0.390 ± 0.15
		Partial (n = 1)	0.544 ± 0.14	0.953 ± 0.05	0.681 ± 0.11	0.898 ± 0.11	0.120 ± 0.03	0.208 ± 0.05	0.264 ± 0.07	0.982 ± 0.03	0.410 ± 0.08
		Partial (n = 2)	0.528 ± 0.13	0.958 ± 0.04	0.670 ± 0.11	0.903 ± 0.10	0.118 ± 0.03	0.205 ± 0.05	0.267 ± 0.06	0.983 ± 0.03	0.415 ± 0.07
Llama-2-70B Chat	5	Exact (n = 0)	0.578 ± 0.15	0.965 ± 0.03	0.709 ± 0.12	0.911 ± 0.13	0.177 ± 0.06	0.291 ± 0.10	0.253 ± 0.11	0.970 ± 0.08	0.388 ± 0.14
		Partial (n = 1)	0.584 ± 0.12	0.965 ± 0.03	0.719 ± 0.10	0.923 ± 0.09	0.170 ± 0.05	0.283 ± 0.07	0.257 ± 0.04	0.976 ± 0.06	0.405 ± 0.05
		Partial (n = 2)	0.575 ± 0.12	0.961 ± 0.04	0.711 ± 0.09	0.917 ± 0.09	0.158 ± 0.05	0.265 ± 0.07	0.267 ± 0.04	0.985 ± 0.04	0.419 ± 0.04