An Exploratory Evaluation of Large Language Models Using Empirical Software Engineering Tasks

Wenjun Liang
Nanjing University of Aeronautics and Astronautics
Nanjing, China
wenjun0418@nuaa.edu.cn

Guanping Xiao*
Nanjing University of Aeronautics and Astronautics
Nanjing, China
gpxiao@nuaa.edu.cn

ABSTRACT

In empirical software engineering (EMSE), various activities require human participation, such as data collection, processing, analysis, and comprehension. On one hand, these processes are time-consuming and labor-intensive. On the other hand, human participation may introduce bias. With the rise of large language models (LLMs) like ChatGPT, the potential for these models to enhance productivity has become apparent. However, the auxiliary capabilities and effectiveness of LLMs in EMSE tasks have rarely been explored. To fill this gap, in this paper, we evaluate the performance of LLMs by using scenarios of human participation in EMSE tasks, i.e., EMSEBench. We conduct replication experiments using four LLMs (ChatGPT4.0, ERNIE Bot4.0, Gemini3.0, and ChatGLM4.0), evaluating the difference in performance across seven scenarios collected from papers published in top SE venues. In the experiments, we perform three types of prompts, i.e., zero-shot, one-shot, and optimized one-shot. Besides, we leverage the concept of multi-agent workflow to explore the performance improvement and limitations of LLMs. Our study summarizes six findings, which facilitate the understanding of the auxiliary of LLMs in EMSE tasks.

CCS CONCEPTS

• **General and reference** → **Evaluation**; **Empirical studies**; **Reliability**.

KEYWORDS

large language models, evaluation benchmark, empirical software engineering tasks

ACM Reference Format:
Wenjun Liang and Guanping Xiao. 2024. An Exploratory Evaluation of Large Language Models Using Empirical Software Engineering Tasks. In *15th Asia-Pacific Symposium on Internetware (Internetware 2024), July 24–26, 2024, Macau, Macao.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3671016.3674821

*Guanping Xiao is the corresponding author.

Internetware 2024, July 24–26, 2024, Macau, Macao

ACM ISBN 979-8-4007-0705-6/24/07
https://doi.org/10.1145/3671016.3674821

1 INTRODUCTION

Large language models (LLMs) are considered one of the crucial pathways toward artificial general intelligence (AGI) [31]. On November 30, 2022, OpenAI introduced ChatGPT (Chat Generative Pre-Trained Transformer) [23], which reached 100 million active users within two months. Over the past year, the impact of LLMs has been truly transformative, revolutionizing academic research and various industrial applications.

One of the key factors making current LLMs both usable and popular is the alignment technique. Alignment refers to the process of ensuring that the behavior of LLMs aligns with human values and preferences [20]. Based on this, scientifically evaluating the performance of LLMs has become a hot research topic. Research on LLM evaluation aims to increase transparency and provide a more comprehensive assessment of these models, further providing improvement guidelines.

To evaluate LLMs, several benchmarks have been proposed, such as SciEval [29], SciBench [30], JEEBench [1], AGIEval [37], C-Eval [12], and M3KE [18]. These benchmarks adopt diverse evaluation scenarios and metric dimensions, considering a broader range of model application contexts. Metrics have evolved from single metric to multi-dimensional metrics, including reliability, safety, fairness, resistance to misuse, explainability and reasoning, adherence to social norms, and robustness [20].

Due to the capability of LLMs in code understanding and generation, many LLM benchmarks built upon code-related tasks in software engineering (SE) have been released like HumanEval [4], MBPP [2], HumanEval$^+$ [19], and EvoEval [32]. However, few have considered evaluating LLMs from the perspective of replicating human participation scenarios in empirical software engineering (EMSE) tasks, such as manual labeling, analysis, and knowledge comprehension and creation [27].

To fill this gap, in this paper, we introduce EMSEBench, containing human participation scenarios and corresponding datasets from EMSE tasks, to evaluate LLMs. We first collect seven human participation scenarios (e.g., bug classification) and their datasets from the top SE venues (i.e., ICSE, FSE, and ASE) over the past three years (21–23). Then, we design different prompts using the collected scenarios to evaluate LLMs (i.e., ChatGPT3.5/4.0 [23], Gemini3.0 [9], ERNIE Bot4.0 [3], and ChatGLM4.0 [36]) in both single-agent and multi-agent workflows. Our work mainly focuses on answering the following two research questions (RQs).

RQ1. *Can empirical software engineering tasks evaluate the performance of LLMs?* In this RQ, we use EMSEBench to evaluate four LLMs (ChatGPT4.0, Gemini3.0, ERNIE Bot4.0, and ChatGLM4.0) with three types of prompts (i.e., zero-shot, one-shot, and optimized one-shot) in a single-agent workflow. **Main Answer.** EMSE tasks significantly distinguish the performance of LLMs: ChatGPT4.0 and ChatGLM4.0 achieve the highest performance with no hallucinations, while ERNIE Bot4.0 and Gemini3.0 both exhibit hallucinations with lower performance.

RQ2. *Can multi-agent workflow improve the performance of LLMs in empirical software engineering tasks?* To fully leverage LLMs' ability, the multi-agent workflow is a promising approach to achieve automated LLM utilization in real-world applications [7]. Therefore, this RQ simulates the human participation process in conducting EMSE tasks by creating three agents with different roles to perform the evaluated tasks using ChatGPT3.5/4.0 and ChatGLM4.0. **Main Answer.** The multi-agent workflow can indeed improves the performance of LLMs.

In this paper, we make the following contributions:

- **Benchmark.** We propose EMSEBench[1], the first benchmark using EMSE tasks for evaluating LLMs.
- **Analysis.** We comprehensively evaluate LLMs on EMSEBench, analyzing different factors impacting the performance of LLMs and their limitations.
- **Findings.** We summarize six findings from our exploratory study, which can facilitate the evaluation of LLMs.

2 DATA COLLECTION AND EXPERIMENT SETUP

2.1 Data Collection

Identifying Keywords for Human Participation Scenarios. Based on our research experience, we first carefully review human participation scenarios collected from specific papers with "Empirical Study" in the title from the top three SE venues, i.e., ICSE, FSE, and ASE, from the past three years 2021 to 2023. Intuitively, the word "manual" is chosen as one candidate of the keyword list. After reading a few papers, terms such as "label", "analyze", and "filter" are added to the list. However, after identifying a sufficient number of human participation scenarios, we find that the aforementioned verb-based keywords are not effective as they represent actions that could be performed either manually or automatically by code or tools, making them neutral terms. After discussion, we decided to retain only "manual" as the final keyword.

Requirements for LLM-Reproducible Human Participation Scenarios. The experimental process described in the paper must be detailed enough, and the authors must provide specific steps or guidelines to ensure we have sufficient information to reproduce the experiments using LLMs. We think the following four types of scenarios cannot be performed by LLMs.

- **Type 1.** The authors only describe what they did (what) rather than how they did it (how).
- **Type 2.** The authors introduce the experimental process through examples (for example) without sufficient details.
- **Type 3.** The experiment is mentioned only to illustrate the advantages of automation (automated) without details.
- **Type 4.** The experimental process is well-detailed but deemed unreplicable by LLMs (e.g., using tools unavailable to LLMs or requiring a questionnaire involving multiple participants).

[1]EMSEBench: https://github.com/EMSEBench.

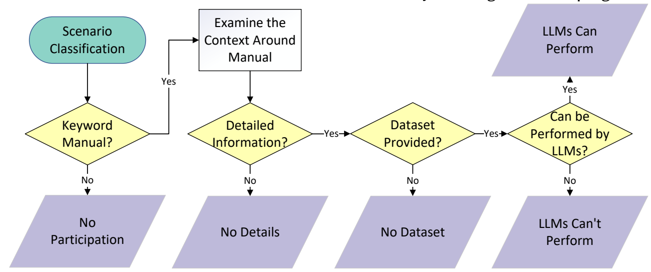


Figure 1: Human-involved scenario classification process.

Classification of Scenarios. The specific classification process is illustrated in Fig. 1. Based on the determined keyword and reproducibility requirements, the classification steps for marking scenarios are as follows: **Step 1.** We search for the keyword "manual" (including "manually") in the papers published in the top three SE venues (ICSE, FSE, and ASE) from 2021 to 2023. Content where the authors explicitly indicate manual participation will be recorded. **Step 2.** For paragraphs identified by the keyword "manual", determine if the manual participation parts provide detailed information for reproduction by LLMs. **Step 3.** If a manual participation experiment meets our requirements, we will search for the corresponding original data and experimental results. Only those providing both process and data will be recorded. **Step 4.** Finally, we check whether the manual participation scenario can be performed by LLMs.

We classify the scenarios as follows:

- **LLM-Reproducible.** The paper includes at least one human participation experiment that meets our reproducibility requirements and provides related data.
- **No Participation.** The paper does not mention human participation, typically evidenced by the absence of the keyword "manual" or mentions of it without relevant content.
- **No Details.** The paper mentions human participation but does not provide sufficient detailed information.
- **No Dataset.** The paper mentions human participation and meets our reproducibility requirements but does not provide the corresponding experimental data.
- **LLM-Unreproducible.** The paper mentions human participation, but the process does not meet our reproducibility requirements. For example, requiring external tools to manually perform the experiments.

Labeling a scenario as "LLM-Unreproducible" is straightforward and less prone to error. Thus, the first round of screening, labeling "No Participation" and "No Dataset" is completed by one author. For the remaining categories, the second round of screening is jointly conducted by two authors, ensuring a scenario is marked "LLM-Reproducible" only if both agree. For other categories, the final result is based on a discussion to make a consensus.

Dataset Construction Results. After classification, we have collected seven target scenarios, along with their corresponding datasets for experiments. Details of the seven scenarios are shown in Table 1. The data type is mainly natural language. For example, Stack Overflow (SO) or GitHub post content, GitHub commit messages, and live-chat transcripts. If the data content exceeds the token limit for a single transmission in LLMs (ChatGLM4.0), we truncate texts to the token limit to ensure consistency in our experiments, which may increase the difficulty of the task for LLMs.

Table 1: Details of Experimental Scenarios

No.	Data Type	Data Source	Data Length	Experiment Content
1	Natural Language	SO/GitHub Issue	Long	Deployment Fault Classification [5]
2	Programming Language	Software System Log	Medium	Log Message and Location Type Classification [17]
3	Natural Language	GitHub Issue	Long	IoT Bug Classification [22]
4	Natural Language	Community Live-chat Log	Short	Live-chat Log Classification [26]
5	Natural Language	SO Issue Post Title	Short	Post Title Classification [15]
6	Natural Language	GitHub Commit Message	Medium	Commit Message Classification [16]
7	Natural Language	TensorFlow.js Issue	Long	JavaScript Fault Classification [24]

RG **You**

Suppose you are a software development engineer.In community-based software development, you need to rely on live-chatting transcripts to discuss emergent bugs/errors you encounter in your daily development tasks.

Now you need to classify the sentences I provide you into:observed behaviors (OB), expected behaviors (EB), steps to reproduce the bug (SR) and others.

tips: "_code_" means code snippet, "_eou_" means the end of a sentence and "_version_" means the version number of a app.

Just give your answer, no explanation required.
If you understand everything I said, please answer Understand.Then I would send the content of live-chatting transcripts.

Figure 2: Example of zero-shot prompt.

2.2 Experiment Setup

Selection of LLMs Under Test. In this study, we refer to the April 2024 SuperCLUE [34], a Chinese LLM ranking, for selecting LLMs. To ensure significant differences in experimental results, we select LLMs with non-adjacent rankings. Considering scores and popularity, we choose ChatGPT4.0 [23] (ranked high) and Gemini3.0 [9] (ranked low), and ERNIE Bot4.0 [3] and ChatGLM4.0 [36] from China (ranked between ChatGPT4.0 and Gemini3.0).

Random Sampling of Test Samples. To ensure the reproducibility and randomness of experimental results, we use a fixed random seed for data sampling. Specifically, during data processing, we initialize a random number generator and set a fixed random seed, making the data sampling process deterministic and ensuring comparisons across different experiments are based on the same random sampling. For the data corresponding to the collected human participation scenarios, we randomly generate about 10 experimental data samples per scenario. According to the categories given in the original paper, we extract the corresponding number of data samples as experimental samples (one sample per category) and provide them to LLMs in one-shot and optimized one-shot prompts.

Prompt Design. For each scenario, we need to design a prompt related to the experimental content of the original paper to ensure that our experiments align with the original experimental content. Prompts are divided into three folds: *zero-shot*, *one-shot*, and *optimized one-shot* prompts, following the thumb of rules in prompt engineering [6, 14]. To facilitate understanding, we chose scenario 4 in Table 1 as an example to illustrate our prompt design in a single-agent experimental environment.

Fig. 2 shows an example of a zero-shot prompt for scenario 4 in Table 1. The first paragraph sets the LLM's role (software development engineer) and explains the task background (solving daily development issues based on chat records). The second and third paragraphs clarify the task content (classifying chat records) and provide specific meanings for labels (i.e., OB, EB, and SR), ensuring the validity of the replication experiment. The fourth paragraph specifies the output format, instructing the LLM to reply with only the label results without additional explanations. The fifth paragraph specifies the response method as replying "Understand"

RG **You**

Suppose you are a software development engineer.In community-based software development, you need to rely on live-chatting transcripts to discuss emergent bugs/errors you encounter in your daily development tasks.

Now you need to classify the sentences I provide you into:observed behaviors (OB), expected behaviors (EB), steps to reproduce the bug (SR) and others.

To help you understand these four categories, I will provide you with an example of each category.
There are examples.
"i have added all the required jars in my eclipse project but still i am getting this error. _eou_" observed behaviors (OB)
"when trying to edit or save desired capabilities, _eou_" expected behaviors (EB)
"i have jdk version _version_ _eou_" steps to reproduce the bug (SR)
"hi have a problem with ios 13 device. _eou_" Others

tips: "_code_" means code snippet, "_eou_" means the end of a sentence and "_version_" means the version number of a app.

Just give your answer, no explanation required.
If you understand everything I said, please answer Understand.Then I would send the content of live-chatting transcripts.

Figure 3: Example of one-shot prompt.

RG **You**

Suppose you are an expert in the field of large language model (LLM) prompt engineering. Now you are required to help a software development engineer optimize his prompt words so that the large language model can better understand his needs.

His prompt words are as follows.The content of the prompt word is enclosed in brackets:

(Suppose you are a software development engineer.In community-based software development, you need to rely on live-chatting transcripts to discuss emergent bugs/errors you encounter in your daily development tasks.

Now you need to classify the sentences I provide you into:observed behaviors (OB), expected behaviors (EB), steps to reproduce the bug (SR) and others.

To help you understand these four categories, I will provide you with an example of each category.
There are examples.
"i have added all the required jars in my eclipse project but still i am getting this error. _eou_" observed behaviors (OB)
"when trying to edit or save desired capabilities, _eou_" expected behaviors (EB)
"i have jdk version _version_ _eou_" steps to reproduce the bug (SR)
"hi have a problem with ios 13 device. _eou_" Others

tips: "_code_" means code snippet, "_eou_" means the end of a sentence and "_version_" means the version number of a app.

Just give your answer, no explanation required.
If you understand everything I said, please answer Understand.Then I would send the content of live-chatting transcripts.)

Figure 4: Example of prompt optimization process.

to prevent unnecessary sentences. Since this is a zero-shot prompt example, no sample example is provided.

Compared to the zero-shot prompt example, the one-shot prompt example, as depicted in Fig. 3, adds one sample for each category (highlighted in red boxes) and reminds the LLM with the phrase "There are examples." The one-shot prompt is expected to perform better than the zero-shot prompt because less effort in prompt crafting typically means less effort from the LLM. Moreover, to compare the prompt optimization by the LLM itself, we design another type of one-shot prompt, i.e., the optimized one-shot prompt. The optimization process for the one-shot prompt is presented in Fig. 4. To successfully optimize the one-shot prompt, we set the LLM's role as a prompt engineering expert and clearly separate the prompt to be optimized using delimiters (i.e., brackets), preventing the LLM from confusing the two parts (highlighted in red boxes in Fig. 4).

Fig. 5 illustrates an example of the optimized one-shot prompt for scenario 4. The optimized prompt is more structured, with all

You

Title: Classify Sentences from Software Development Live-Chat Transcripts

Task Description:

As a software development engineer, you often rely on live-chat transcripts to discuss and troubleshoot emergent bugs and errors encountered during your development tasks. Your task is to classify the provided sentences from these transcripts into four distinct categories: Observed Behaviors (OB), Expected Behaviors (EB), Steps to Reproduce the Bug (SR), and Other.

Instructions:

Observed Behaviors (OB): Classify sentences that describe actual behaviors or errors experienced. For example: "I have added all the required jars in my Eclipse project but still I am getting this error. eou"
Expected Behaviors (EB): Classify sentences that describe the expected or intended behaviors of the software. For example: "When trying to edit or save desired capabilities, eou"
Steps to Reproduce the Bug (SR): Classify sentences detailing specific steps or conditions needed to replicate the bug. For example: "I have JDK version version eou"
Other: Classify sentences that do not fit into the above categories or are general queries. For example: "Hi, I have a problem with iOS 13 device. eou"

Notes:

"code" represents a code snippet.
"eou" denotes the end of a sentence.
"version" indicates a software version number.

Response Format:

Simply classify each sentence without the need for an explanation.

Confirmation:

If you understand the instructions, please reply with "Understand". Upon confirmation, I will send you the content of the live-chatting transcripts.

Figure 5: Example of optimized One-shot prompt.

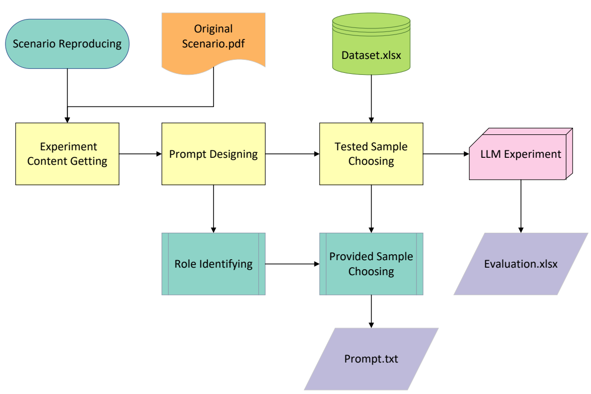


Figure 6: Single-agent evaluation process.

components highlighted as subheadings (such as Task). In practice, optimized one-shot prompts from LLMs often lose the original category meanings or sample examples. Besides, specific response format requirements for the LLM may also be lost. If these critical components are missing, the replication experiment may not proceed. In this case, we adjust the optimized one-shot prompts.

3 RQ1: PERFORMANCE OF LLMS ON EMSEBENCH

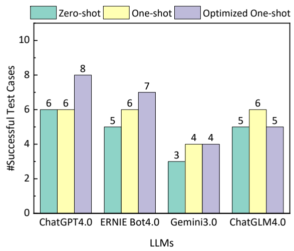
3.1 Evaluation Process

Fig. 6 shows the evaluation process of the single-agent experiment. First, for the scenarios to be tested, we review the corresponding paper to clarify the original experimental content and process, transforming it into our prompt's "Task" and "Content" sections. For the "Role" section, we extract the purpose from the context of the human-involved experiment part of the original paper and ask the LLM to suggest a reasonable role based on this information. The "Specified Response" section is determined by our output requirements. At this point, the zero-shot prompt design is complete.

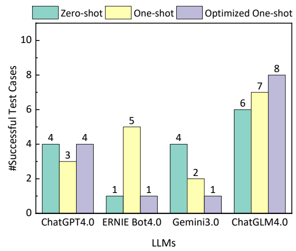
Next, using the designed random number generator, we extract the corresponding number of data samples from the dataset based on the category count of the original scenario experiments and add them to the one-shot prompt. Note that in our default evaluation process, all content of the prompt is sent in the first message. However, if the sample length (actually the length of the data) is too long to be sent in a single transmission (exceeding the LLM's token limit), we send the one-shot prompt in multiple transmissions. In the first transmission, the content is similar to the zero-shot prompt, but with an additional instruction in the "Specified Response" section, such as, "To deepen your understanding of the category meanings, we provide a sample for each category. Please reply 'Understand' to indicate you understand my request." This "Understand" response specification prevents the LLM from misunderstanding the prompt and replying with unnecessary content. If multiple transmissions are needed to send the samples, we use the same method to maintain coherence between the prompts, avoiding unexpected replies. After all samples are sent, we modify the "Specified Response" section to notify the LLM to start the classification experiment. At this point, the one-shot prompt design is complete.

Finally, we provide the designed one-shot prompt to the LLM (whose role is preset as an "expert in the field of large language model prompt engineering"), and obtained the optimized one-shot prompt (this process is called the "optimization process"). If the one-shot prompt content exceeds the LLM's token limit and is sent in multiple transmissions, the optimized one-shot prompt would also follow the same transmission process. The optimized one-shot prompt must maintain the same experimental content as the original experiment, with no changes in the description of categories. If the optimized one-shot prompt removes the specific descriptions of the categories (leaving only the category names) or modifies the content, we will not accept the optimized result. We will modify the prompts in the "optimization process" instructing the LLM to retain the original experimental content and optimize again until the optimized one-shot meets our expected results. At this point, the optimized one-shot prompt design is complete.

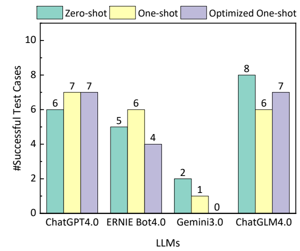
It should be noted that each scenario's replication experiment should involve at least four sessions with the LLM, corresponding to "zero-shot prompt experiment", "one-shot prompt experiment", "optimization process", and "optimized one-shot prompt experiment". This means that the replication experiment process for the three types of prompts and the one-shot prompt optimization process are independent, with one session in the browser for each process, ensuring the independence of the replication experiments. Mixing these four processes in one session would result in an increasingly rich context for later prompts, which is unfair to other prompts. Additionally, if necessary, a fifth session (actually the first in order) can be added to provide the experimental content to the LLM and ask it for a suitable role for the scenario (used in the "Role" section of the prompts). However, in most cases, defining the LLM's role as "software developer" or "an expert in the field of..." is well sufficient.



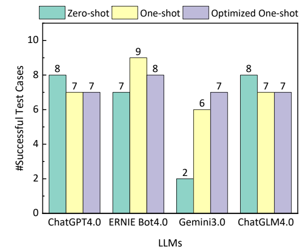
(a) Scenario1 (9 samples)



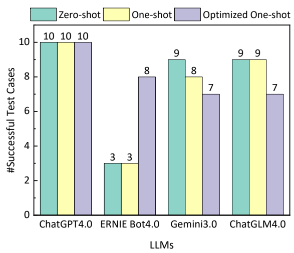
(b) Scenario2 (15 samples)



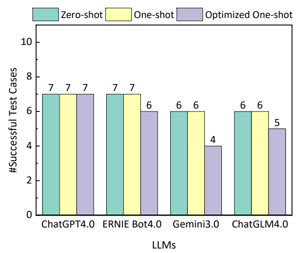
(c) Scenario3 (10 samples)



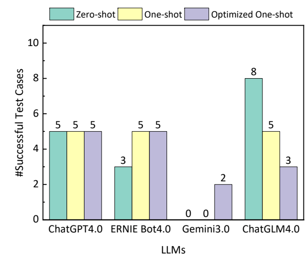
(d) Scenario4 (10 samples)



(e) Scenario5 (10 samples)



(f) Scenario6 (10 samples)



(g) Scenario7 (10 samples)

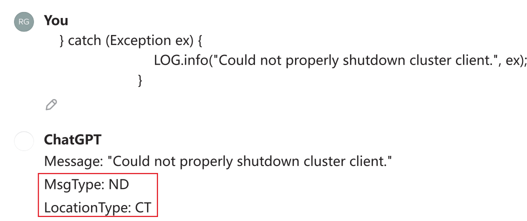
Figure 7: Comparison of LLMs on EMSEBench in single-agent workflow.

3.2 Results Analysis

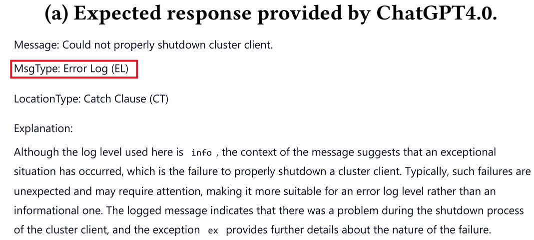
3.2.1 Performance of LLMs on EMSEBench in Single-agent Workflow. We first present the replication experiment results of the seven scenarios in the form of bar charts, as shown in Fig. 7. Since the main tasks of these seven scenarios are classification tasks, "#Successful Test Cases" implies that the number of classification results given by the LLMs that match the original paper's results.

Scenario 1. *Classification of Deployment Faults* [5]. The experiment involves reading SO/GitHub post content and classifying deep learning mobile application deployment faults. The original paper provides specific classification methods down to subcategories, but due to insufficient detail, we use the results of major categories as a reference. The experimental data is long natural language post content, requiring truncation when sending one-shot and optimized one-shot prompts. The number of samples evaluated is nine. From Fig. 7a, ChatGPT4.0 performs the best, followed by ERNIE Bot4.0 and ChatGLM4.0, all three LLMs with accuracy over 50%. Gemini3.0 performs the worst and does not reach 50% accuracy. ERNIE Bot4.0 exhibits hallucinations using the optimized one-shot prompt, providing non-existent categories.

Scenario 2. *Classification of Log Information* [17]. The experimental data are log statements in code snippets, with medium-length code. The task is to extract log messages and classify them by type. The original paper classifies both message type and location type, so we considered results matching both types as correct. The number of samples is 15, with ChatGLM4.0 achieving the highest accuracy, close to 50%, followed by ChatGPT4.0, Gemini3.0, and ERNIE Bot4.0. ENRIE Bot4.0 exhibits hallucinations again in zero-shot and optimized one-shot prompts, resulting in very low accuracy. For example, the expected message type is ND for the tested log statement (Fig. 8a), but ERNIE Bot4.0 provides a non-existing type, i.e., EL (Fig. 8b). By contrast, most samples are mistakenly classified by Gemini3.0, leading to low performance. Notably, Gemini3.0 provides additional explanations despite the prompt clearly



(a) Expected response provided by ChatGPT4.0.



(b) Hallucination provided by ERNIE Bot4.0.

Figure 8: Example of expected response and hallucination.

stating "No additional explanation is needed" unlike ChatGPT4.0 and ChatGLM4.0, which adhere to the prompt requirements.

Scenario 3. *Classification of IoT Bugs* [22]. The experimental data are issue content, similar to Scenario 1, involving reading GitHub issue content and classifying IoT bug types. The number of samples is 10 in this scenario. ChatGPT4.0 and ChatGLM4.0 have similar accuracy, around 70%, with ERNIE Bot4.0 next, and Gemini3.0 performing the worst. Gemini3.0 exhibits hallucinations in all prompt experiments, leading to very low overall accuracy.

Scenario 4. *Classification of Live-chat Transcripts* [26]. The experimental data are short sentences of chat records, related to development issues and errors, classified by specific content. The number of samples is 10, with ChatGPT4.0, ERNIE Bot4.0, and ChatGLM4.0 performing well, while Gemini3.0 performing moderately. Both ERNIE Bot4.0 and Gemini3.0 exhibit hallucinations, particularly

Gemini3.0 has numerous hallucinations in the zero-shot prompt experiment. Additionally, Gemini3.0 provides unnecessary explanations in responses again.

Scenario 5. *Classification of Post Titles* [15]. The experimental data are short post titles. The task is to classify SO post titles into three types: conceptual questions, how-to questions, and bug-fixing questions. The number of samples is 10. During the evaluation, both ERNIE Bot4.0 and Gemini3.0 exhibit hallucinations, but these are understandable and acceptable. Some titles are in question form (interrogative sentences), and without additional hints, LLMs might assume the user changed the query content, leading to unexpected answers. We provide delimiters, e.g., "The question of SO post is as follows. Remember your task is to categorize the question", to clarify the experimental data and ensure the LLM knows its tasks. However, even with delimiters, ERNIE Bot4.0 and Gemini3.0 still exhibit hallucinations. We find that ChatGPT4.0 with 100% accuracy, followed by ChatGLM4.0 with about 80%. Gemini3.0 performs comparably to ChatGLM4.0 after providing additional prompts, but ERNIE Bot4.0 still has many errors.

Scenario 6. *Classification of Commit Messages* [16]. The experimental data are medium-length commit messages. The task is to classify whether the commit messages contained "Why" and "What" content. Special cases require the LLM to read post content in <link> tags, making the experiment more complex. The number of samples is 10, with ChatGPT4.0, ERNIE Bot4.0, and ChatGLM4.0 performing well. ERNIE Bot4.0 exhibits hallucinations that do not affect the experiment, with correct types but incorrect labels. Gemini3.0 exhibits the same errors as ERNIE Bot4.0 and misinterprets URL information, thinking it needs to access websites, leading to errors.

Scenario 7. *Classification of JavaScript-based DL Faults* [24]. The data are long post content divided into title, body, and comments by the original authors. The task is to classify fault types in JavaScript-based TensorFlow projects. The number of samples is 10, with ChatGPT4.0, ERNIE Bot4.0, and ChatGLM4.0 performing similarly, with about 50% accuracy. Gemini3.0 could not experiment due to URL information in the post content (despite the experiment not requiring website access).

Overall, ChatGPT4.0 and ChatGLM4.0 have the highest replication accuracy at about 62% (139/222, 138/222), with no hallucinations. ERNIE Bot4.0 follows, with frequent hallucinations leading to 50% (111/222) accuracy. Gemini3.0 had the lowest performance, with hallucinations and URL misinterpretations, at 35.1% (78/222).

> **Finding #1**: In EMSE tasks, ChatGPT4.0 and ChatGLM4.0 have the highest performance with no hallucinations during replication experiments. ERNIE Bot4.0 frequently exhibits hallucinations, leading to lower accuracy than ChatGPT4.0 and ChatGLM4.0. Gemini3.0 exhibits both hallucinations and misinterpretations, with the lowest replication accuracy.

3.2.2 Performance of LLMs with Different Prompts. To evaluate the performance of different prompts in single-agent workflow, we record the highest accuracy LLM/prompt combination for each scenario, as shown in Table 2. For example, in Scenario 1, if the combination (ChatGPT4.0/optimized one-shot) achieves the highest accuracy, the count for this combination is increased by one. The initial value of all counts is zero. Through "Total" in the last row,

Table 2: Statistics of the Combination of Prompt Types and LLMs with the Highest Reproduction Accuracy

LLM/Prompt	Zero-shot	One-shot	Optimized One-shot	Total
ChatGPT4.0	2	2	3	7
ERNIE Bot4.0	1	2	0	3
Gemini3.0	0	0	0	0
ChatGLM4.0	2	0	1	3
Total	5	4	4	13

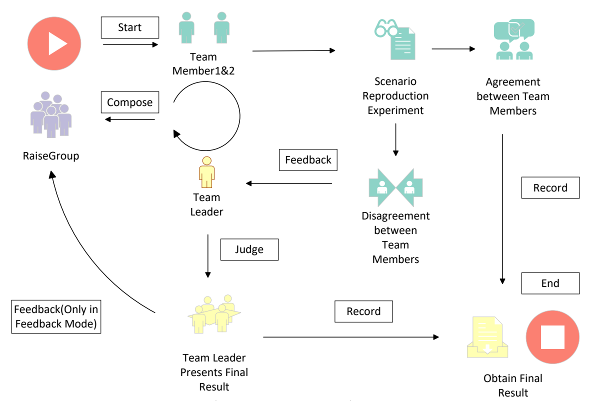


Figure 9: Multi-agent evaluation process.

we can rank the performance of the three prompt types. This table also verifies Finding #1. Notably, in Scenarios 5 and 6 (see Fig. 7e and Fig. 7f), some LLMs (e.g., ChatGPT4.0) achieve the best and consistent accuracy with all three prompts, thus these combinations are counted multiple times, with the final total summing to 13.

It is worth mentioning that the optimization process does not always yield the expected results. The "Role" and "Specified Response" sections in one-shot prompts can affect the LLM's optimization ability. In some cases, we temporarily remove content not related to the original experiment, optimize the remaining parts, and then supplement the removed content. This means the use of optimized one-shot prompts currently relies on human intervention, and their practical significance remains to be further explored.

> **Finding #2**: In EMSE tasks, zero-shot prompts have the highest performance, followed by one-shot and optimized one-shot prompts.

4 RQ2: IMPACT OF MUTI-AGENT WORKFLOW OF LLMS ON EMSEBENCH

4.1 Evaluation Process

Fig. 9 shows the multi-agent workflow process. Given the unique nature of our experimental data, we initialize the multi-agent workflow as a software engineering group, RaiseGroup. The group consists of two team members with the role of software development engineer) and a team leader with the role of quality assurance (QA) engineer. The two members are on equal footing and independently solve the same problem (initiating two separate browser sessions with the LLM) to obtain their results. The decision-making process for the team members' results is as follows:

- If the replication results of both team members (the results of the problem in our experiments) are the same, regardless of whether the result is correct or not, it will be taken as the final classification result without being sent to the team leader. This is based on the rationale that, in real-world research, RG **You**

The RaiseGroup team consists of two software development engineers (team members) and one Quality Assurance (QA) engineer (team leader). You are one of the software development engineers, and you frequently solve your programming needs by reading posts on StackOverflow (SO). While manually reviewing SO posts, you noticed a correlation between the questions and the answers in the posts. For research purposes, your main task now is to categorize the questions in the provided SO posts into the following three categories:
1. conceptual questions: asking for clarifications on a concept.
2. how-to questions: asking for instructions for achieving a task.
3. bug fixing questions: asking for solutions to fix some issues.
Answer Types 1, 2, or 3, and provide your reasons.

Please note that the other team member will perform the same task independently, and both of your answers and reasons will be submitted to the team leader. If your answers differ from those of the other team member, the team leader will evaluate both sets of answers and reasons, and select the most reasonable one as the final result. This final decision will be communicated to the team member whose answer differs, who may then either modify their answer or choose to keep it.
If you understand your role and assignment, please reply with "Understand." After that, you can begin your work.

(a) Team member initial setting.

RG **You**

Your answer differed from that of another team member. After the team leader analyzed the answers and reasons of the two team members, the team leader believed that the other team member's answer was more reasonable, and the team leader gave his reasons.
The reasons for the team leader are as follows (you are team member 1, and another team member is team member 2):

(b) Feedback to team member (only in feedback mode).

Figure 10: Example of multi-agent member prompt.

the team leader's role is to make decisions only when there is a disagreement among team members.

- If the replication results differ, we handle it in two ways. The *first* method, the no-feedback approach, involves sending both team members' results and their reasoning to the team leader (initiating a third session within the experiment). The team leader selects the most reasonable result as the final outcome and provides reasoning. The *second* method, the feedback approach, extends the first method by also sending the team leader's final result and reasoning back to the member whose result was not chosen. The member can then decide whether to update his result to match the team leader's or retain his original result with further justification. However, following the principle of majority rule, this decision does not affect the final outcome. We design both methods to investigate whether feedback from the team leader improves the accuracy of team members' future classifications.

With the multi-agent workflow established, our prompts also need to be adapted accordingly. We keep the original content of the prompts unchanged, maintaining their internal structure. We add a description of RaiseGroup at the beginning of all prompts to introduce the concept of the group to the LLM (making each role aware of each other's existence) and provide task descriptions based on the LLM's role (team member or team leader), as shown in Fig. 10a and Fig. 11a. At the end of all prompts, we include feedback instructions based on the LLM's role. Team members are informed that if there is a disagreement, both classification results will be sent to the team leader for judgment (Fig. 10a). The team leader is informed that the team members are working on the same task and that any conflicting results will be resolved by it (Fig. 11b). If the feedback mode is active, the team leader's final result will be sent back to the team member (see the text with red underline in Fig. 10a), who will then decide whether to update its result (Fig. 10b).

RG **You**

The RaiseGroup team consists of two software development engineers (team members) and one Quality Assurance (QA) engineer (team leader). You are the QA engineer (team leader). Currently, the team members are tasked with categorizing the questions in the given StackOverflow (SO) posts into three categories:
1. conceptual questions: asking for clarifications on a concept.
2. how-to questions: asking for instructions for achieving a task.
3. bug fixing questions: asking for solutions to fix some issues.
Their answers will be provided in the form of category along with reasons.

Please note that both team members are performing the same task independently, and their answers and reasons will be submitted to you. If both team members provide the same answer, then that answer becomes the final result. If their answers differ, you need to evaluate the answers and reasons from both members and choose the most reasonable one as the final outcome. This result will then be communicated to the team member whose answer differs from the chosen one, who may either change their answer or continue to maintain it.

If you understand your role and assignment, please reply with "Understand." After that, you can begin your work.

(a) Team leader initial setting.

RG **You**

The two team members disagreed on the classification of the following title(separated by ## symbols):
#How to create a generic array in Java?#

Team member 1 thinks:
This question belongs to the category of how-to questions. The title "How to create a generic array in Java?" explicitly requests instructions or a method for accomplishing a specific programming task in Java—creating a generic array. This indicates that the user is looking for a step-by-step guide or explanation on how to implement this in their code, typical of how-to queries.

Team member 2 thinks:
This question belongs to the conceptual category. While it appears to be asking "how to" do something, the underlying challenge—creating a generic array in Java—involves understanding Java's type system and the limitations imposed on generics due to type erasure. Therefore, the question is likely to involve explanations on why Java does not allow the creation of generic arrays directly and what conceptual workarounds can be used instead. This requires clarifications on a concept rather than straightforward procedural instructions.

Please choose the one you think is the most reasonable based on the reasons given by the two team members as the final answer.

(b) Disagreement resolution prompt of team leader.

Figure 11: Example of multi-agent leader prompt.

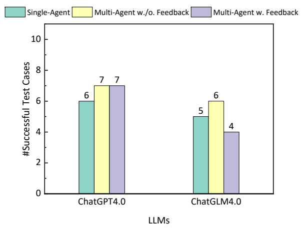
Since our goal is to explore whether the multi-agent workflow improves LLM performance, the type of prompt is not a consideration. We only use the best-performing zero-shot prompt (Finding #2) in the multi-agent experiments and reference the zero-shot prompt results from the single-agent experiments.

Besides, due to the frequent occurrence of hallucinations with ERNIE Bot4.0 and Gemini3.0 in single-agent experiments, which prevent the tests from proceeding, we decide to only use ChatGPT4.0 and ChatGLM4.0 for the multi-agent experiments. For comparison, we only reference the single-agent experiment results of ChatGPT4.0 and ChatGLM4.0. We add content delimiters (e.g., "#") when sending test samples to ensure the experiments ran smoothly.

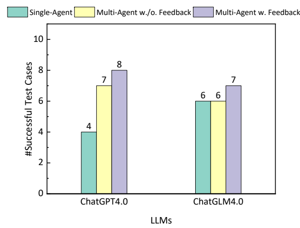
4.2 Results Analysis

4.2.1 Comparison of Performance Between Single-agent and Multi-agent Workflows. We first compare the performance of single-agent and multi-agent workflows across seven scenarios, with the multi-agent workflow divided into no-feedback and feedback modes.

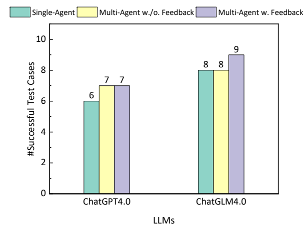
As shown in Fig. 12, considering the two modes in the multi-agent workflow, except for a significant performance difference in Scenarios 1 and 6 with ChatGLM4.0 between no-feedback and feedback modes, the performance is generally consistent across other scenarios, with a discrepancy of about one successful replication. Therefore, we calculate the average of the results from the two modes as the final result for the multi-agent workflow.



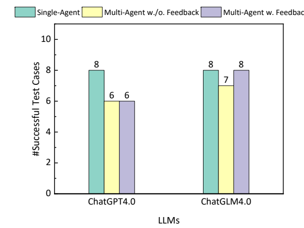
(a) **Scenario1 (9 samples)**



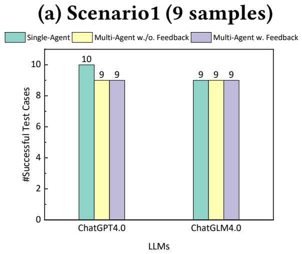
(b) **Scenario2 (15 samples)**



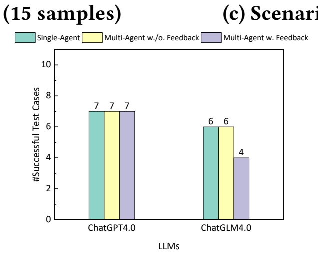
(c) **Scenario3 (10 samples)**



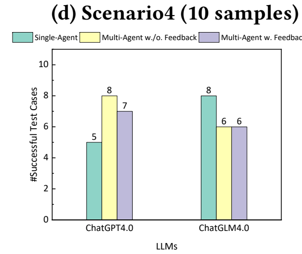
(d) **Scenario4 (10 samples)**



(e) **Scenario5 (10 samples)**



(f) **Scenario6 (10 samples)**



(g) **Scenario7 (10 samples)**

Figure 12: Comparison of LLMs on EMSEBench in multi-agent workflow.

Table 3: Comparison of ChatGPT4.0's Performance Between Single-agent and Multi-agent Workflows

	Scenario1	Scenario2	Scenario3	Scenario4	Scenario5	Scenario6	Scenario7
Single-agent	6	4	6	8	10	7	5
Multi-agent	7	7.5	7	6	9	7	7.5

Table 4: Comparison of ChatGLM4.0's Performance Between Single-agent and Multi-agent Workflows

	Scenario1	Scenario2	Scenario3	Scenario4	Scenario5	Scenario6	Scenario7
Single-agent	5	6	8	8	9	6	8
Multi-agent	5	6.5	8.5	7.5	9	5	6

First, we analyze the results of ChatGPT4.0, as shown in Table 3. In Scenarios 1, 2, 3, and 7, the multi-agent workflow performs better, whereas in Scenarios 4 and 5, the single-agent workflow is superior. The performance is the same in Scenario 6. Overall, the single-agent workflow has a replication accuracy of 62.2% (46/74), while the multi-agent workflow achieves 68.9% (51/74).

For ChatGLM4.0, the results are shown in Table 4. Unlike ChatGPT4.0, the performance is the same in Scenarios 1, 4, and 5, slightly better in the multi-agent workflow for Scenarios 2 and 4, but worse in the remaining scenarios. Overall, the single-agent workflow has a replication accuracy of 67.6% (50/74), while the multi-agent workflow achieves 64.2% (47.5/74).

Combining the results for both LLMs, the single-agent workflow has a replication accuracy of 64.9% (48/74), while the multi-agent workflow has 66.6% (49.25/74).

> **Finding #3**: In EMSE tasks, the multi-agent workflow performs better than the single-agent workflow for ChatGPT4.0 but worse for ChatGLM4.0. Overall, the multi-agent workflow has higher performance than the single-agent workflow.

4.2.2 Relationship Between LLM Consistency and Error-Correction Ability. We observe that the "team leader decision" step in the multi-agent workflow can be seen as a "reflection" mode, leveraging the LLM's error-correction ability through discussions between two agents (two sessions of the same LLM). This mode significantly improves application results in some cases. We evaluate the LLM's self-correction ability by tracking the accuracy of the "team leader" judgments in the multi-agent workflow, as shown in Table 5. For example, in Scenario 2 with ChatGPT 4.0, "1/1" indicates that there are two cases requiring the team leader's judgment in the feedback modes, with one correct and one incorrect decision.

Table 5: The Correctness/Wrongness of the Team Leader's Final Result

	Scenario1	Scenario2	Scenario3	Scenario4	Scenario5	Scenario6	Scenario7
ChatGPT4.0	0/0	1/1	1/2	0/3	1/2	0/1	3/0
ChatGLM4.0	2/4	2/1	1/0	2/1	0/1	0/2	1/2

Note that when both "team members" give incorrect but different results, the "team leader" choosing one of these incorrect results (rather than a new correct one) is also considered an incorrect judgment. While we only require the team leader to choose the most reasonable result, we observe cases where the team leader provided a new correct result, which we consider an indication of the LLM's correction ability.

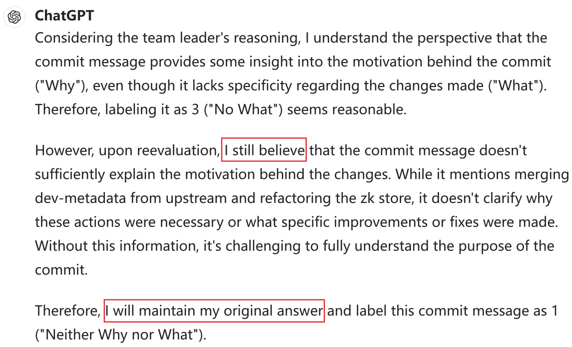
Table 5 shows that ChatGPT4.0 has 6 correct and 9 incorrect judgments, with a correction rate of 40% (6/15). ChatGLM4.0 has 8 correct and 11 incorrect judgments, with a correction rate of 42.1% (8/19). While ChatGPT4.0 performed better overall in the multi-agent workflow, its correction ability is inferior to ChatGLM4.0. Given the number of team leader's judgments, we infer: ChatGPT4.0 has higher consistency than ChatGLM4.0. With both having low self-correction ability, we can conclude that the higher consistency of LLMs leads to better performance.

> **Finding #4**: In EMSE tasks, both ChatGPT4.0 and ChatGLM4.0 have low self-correction ability. ChatGLM4.0 performs slightly better self-correction ability than ChatGPT4.0. When LLM self-correction is insufficient in multi-agent workflow, higher consistency results in better performance.

4.2.3 Relationship Between LLM Consistency and Sycophancy. We note an interesting phenomenon where both ChatGPT4.0 and ChatGLM4.0 "team members" accept the team leader's final result and update their original result, even though the team leader's judgment is often incorrect. This might indicate sycophancy in LLMs, where LLMs unconditionally accept authority figures' requirements [20].

Table 6: Comparison of Inconsistency Between ChatGPT 4.0 and 3.5

.	Scenario1	Scenario2	Scenario3	Scenario4	Scenario5	Scenario6	Scenario7
ChatGPT4.0	0	2	3	3	3	1	3
ChatGPT3.5	5	5*	5	5	2	4*	3


ChatGPT

Considering the team leader's reasoning, I understand the perspective that the commit message provides some insight into the motivation behind the commit ("Why"), even though it lacks specificity regarding the changes made ("What"). Therefore, labeling it as 3 ("No What") seems reasonable.

However, upon reevaluation, I still believe that the commit message doesn't sufficiently explain the motivation behind the changes. While it mentions merging dev-metadata from upstream and refactoring the zk store, it doesn't clarify why these actions were necessary or what specific improvements or fixes were made. Without this information, it's challenging to fully understand the purpose of the commit.

Therefore, I will maintain my original answer and label this commit message as 1 ("Neither Why nor What").

Figure 13: Team member maintains answer in ChatGPT3.5.

We hypothesize that sycophancy might relate to LLM consistency, causing the LLM to yield consistent results over multiple attempts. We conduct supplementary experiments with ChatGPT3.5, as shown in Table 6. For instance, in Scenario 2, "2" for ChatGPT4.0 indicates that out of 20 experiments (10 samples, two feedback modes), there are two instances of disagreement among the team members.

Table 6 shows that ChatGPT4.0 generally has higher consistency than ChatGPT3.5, except in Scenario 5. During the ChatGPT3.5 experiments, in scenarios 2 and 6 (marked with "*") team members retain their opinions (non-sycophancy, see Fig. 13). While team member opinion retention does not affect our final result in replication experiments, it could lower LLM performance in multi-level feedback workflows. Hence, we believe sycophancy is more likely in higher consistency LLMs, affecting performance. Future LLM alignment training should carefully consider this issue.

> **Finding #5**: In EMSE tasks, sycophancy may reduce LLM performance in multi-level feedback workflows, likely appearing in higher consistency LLMs. Reduced sycophancy may lead to more reasonable results, thereby improving performance.

4.2.4 Comparison of Performance Between ChatGPT4.0 with Single-agent and ChatGPT3.5 with Multi-agent Workflows. ChatGPT4.0 is the upgraded version of ChatGPT3.5, with numerous benchmarks indicating superior performance [34]. To further validate the improvement of multi-agent workflow on LLM performance, we compare single-agent ChatGPT4.0 with multi-agent ChatGPT3.5.

Note that for single-agent ChatGPT4.0, we only consider zero-shot prompt results. For multi-agent ChatGPT3.5, we average the two workflow modes. As shown in Fig. 14, single-agent ChatGPT4.0 has a replication accuracy of 62.2% (46/74), while multi-agent ChatGPT3.5 has 60.1% (44.5/74). We conclude that multi-agent ChatGPT3.5 matches the performance of ChatGPT4.0, implying that multi-agent workflow can indeed improve LLM performance.

> **Finding #6**: In EMSE tasks, multi-agent workflows significantly improve LLM performance. Multi-agent ChatGPT3.5 can achieve a similar performance to ChatGPT4.0.

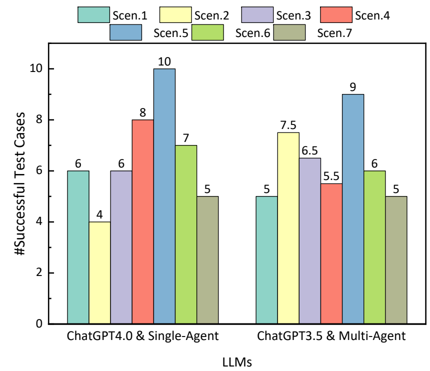


Figure 14: Comparison of ChatGPT4.0 and ChatGPT3.5 with different workflows.

5 THREATS TO VALIDITY

Threats to Internal Validity. The main threats to internal validity come from the data collection and experiment setup. We defined rigorous criteria to filter papers to select human participation scenarios. Regarding replicating human participation scenarios by LLMs, we strictly followed the definitions and descriptions provided by the selected papers to refine the prompts. Moreover, since we used the web platforms to access and evaluate the LLMs, our results are only valid when we conducted the evaluations.

Threats to External Validity. The main threat to external validity lies in the generalization of our finding results. In our exploratory study, we tested seven scenarios with limited samples on four LLMs. This limited dataset may introduce bias into the findings. To mitigate this threat, we plan to evaluate more scenarios on more LLMs in future work.

Threats to Construct Validity. The threat to construct validity relates to the metric used for evaluation. In our study, we calculated the accuracy, the ratio of corrected responses to the total tested samples, to evaluate LLMs.

6 RELATED WORK

LLM Evaluation Benchmarks. Over the past year, many LLM evaluation benchmarks have been introduced, e.g., SciEval [29], SciBench [30], JEEBench [1], AGIEval [37], C-Eval [12], and M3KE [18]. Huang *et al.* [12] proposed C-EVAL, the first comprehensive Chinese evaluation suite designed to assess advanced knowledge and reasoning abilities of foundational models in a Chinese context. C-EVAL includes multiple-choice questions at four difficulty levels: middle school, high school, college, and professional.

LLMs in SE. Recently, LLMs have been applied to many software engineering tasks such as automated program repair [8, 10, 11, 13, 33, 35] and software testing [21, 25, 28]. To evaluate LLMs in code understanding and generation, many code-related benchmarks also have been introduced, e.g., HumanEval [4], MBPP [2], HumanEval^{+} [19], and EvoEval [32]. Compared to existing work, our study presents the first evaluation of LLMs using human participation scenarios in EMSE tasks.

7 CONCLUSION

In this paper, we conducted an exploratory investigation to compare the performance of different LLMs using EMSEBench, human participation tasks in EMSE. We used seven human participation scenarios and related data to evaluate the performance of LLMs, including ChatGPT4.0, ERNIE Bot4.0, Gemini3.0, and ChatGLM4.0 with three types of prompts, i.e., zero-shot, one-shot, and optimized one-shot prompts. Besides, we analyzed multi-agent workflow technique to explore its actual improvement in LLM performance. We believe this research can facilitate the understanding of the auxiliary role and effectiveness of LLMs in EMSE research.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable comments and suggestions. This work was supported in part by National Natural Science Foundation of China under Grant 62002163 and Natural Science Foundation of Jiangsu Province under Grant BK20200441.

REFERENCES

[1] Daman Arora, Himanshu Gaurav Singh, et al. 2023. Have llms advanced enough? a challenging problem solving benchmark for large language models. *arXiv preprint arXiv:2305.15074* (2023).

[2] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732* (2021).

[3] Baidu. 2024. ERNIE Bot. https://yiyan.baidu.com/welcome. (Accessed on 05/18/2024).

[4] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[5] Zhenpeng Chen, Huihan Yao, Yiling Lou, Yanbin Cao, Yuanqiang Liu, Haoyu Wang, and Xuanzhe Liu. 2021. An empirical study on deployment faults of deep learning based mobile applications. In *Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 674–685.

[6] Yihong Dong, Xue Jiang, Zhi Jin, and Ge Li. 2023. Self-collaboration code generation via chatgpt. *arXiv preprint arXiv:2304.07590* (2023).

[7] Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. 2023. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325* (2023).

[8] Zhiyu Fan, Xiang Gao, Martin Mirchev, Abhik Roychoudhury, and Shin Hwei Tan. 2023. Automated repair of programs from large language models.. In *Proceeding of the 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1469–1481.

[9] Google. 2024. Gemini - chat to supercharge your ideas. https://gemini.google.com/. (Accessed on 05/18/2024).

[10] Soneya Binta Hossain, Nan Jiang, Qiang Zhou, Xiaopeng Li, Wen-Hao Chiang, Yingjun Lyu, Hoan Nguyen, and Omer Tripp. 2024. A deep dive into large language models for automated bug localization and repair. *arXiv preprint arXiv:2404.11595* (2024).

[11] Kai Huang, Xiangxin Meng, Jian Zhang, Yang Liu, Wenjie Wang, Shuhao Li, and Yuqing Zhang. 2023. An empirical study on fine-tuning large language models of code for automated program repair. In *Proceedings of the 2023 IEEE/ACM 38th International Conference on Automated Software Engineering (ASE)*. IEEE, 1162–1174.

[12] Yuzhen Huang, Yuzhuo Bai, Zhihao Zhu, Junlei Zhang, Jinghan Zhang, Tangjun Su, Junteng Liu, Chuancheng Lv, Yikai Zhang, Yao Fu, et al. 2024. C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models. *Advances in Neural Information Processing Systems* 36 (2024).

[13] Nan Jiang, Kevin Liu, Thibaud Lutellier, and Lin Tan. 2023. Impact of code language models on automated program repair. In *Proceedings of the 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1430–1442.

[14] Jonathan Kemper. 2024. ChatGPT Guide: Use these prompt strategies to maximize your results. https://the-decoder.com/chatgpt-guide-prompt-strategies/#google_vignette. (Accessed on 05/18/2024).

[15] Bonan Kou, Muhao Chen, and Tianyi Zhang. 2023. Automated summarization of stack overflow posts. In *Proceedings of the 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1853–1865.

[16] Jiawei Li and Iftekhar Ahmed. 2023. Commit message matters: Investigating impact and evolution of commit message quality. In *Proceedings of the 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 806–817.

[17] Zhenhao Li, Heng Li, Tse-Hsun Chen, and Weiyi Shang. 2021. Deeplv: Suggesting log levels using ordinal based neural networks. In *Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1461–1472.

[18] Chuang Liu, Renren Jin, Yuqi Ren, Linhao Yu, Tianyu Dong, Xiaohan Peng, Shuting Zhang, Jianxiang Peng, Peiyi Zhang, Qingqing Lyu, et al. 2023. M3ke: A massive multi-level multi-subject knowledge evaluation benchmark for chinese large language models. *arXiv preprint arXiv:2305.10263* (2023).

[19] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2024. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems* 36 (2024).

[20] Yang Liu, Yuanshun Yao, Jean-Francois Ton, Xiaoying Zhang, Ruocheng Guo Hao Cheng, Yegor Klochkov, Muhammad Faaiz Taufiq, and Hang Li. 2023. Trustworthy LLMs: a survey and guideline for evaluating large language models' alignment. *arXiv preprint arXiv:2308.05374* (2023).

[21] Zhe Liu, Chunyang Chen, Junjie Wang, Mengzhuo Chen, Boyu Wu, Xing Che, Dandan Wang, and Qing Wang. 2024. Make llm a testing expert: Bringing human-like interaction to mobile gui testing via functionality-aware decisions. In *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering*. 1–13.

[22] Amir Makhshari and Ali Mesbah. 2021. IoT bugs and development challenges. In *Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 460–472.

[23] OpenAI. 2024. ChatGPT | OpenAI. https://openai.com/chatgpt/. (Accessed on 05/18/2024).

[24] Lili Quan, Qianyu Guo, Xiaofei Xie, Sen Chen, Xiaohong Li, and Yang Liu. 2022. Towards understanding the faults of javascript-based deep learning systems. In *Proceedings of the 2022 IEEE/ACM 37th International Conference on Automated Software Engineering (ASE)*. 1–13.

[25] Gabriel Ryan, Siddhartha Jain, Mingyue Shang, Shiqi Wang, Xiaofei Ma, Murali Krishna Ramanathan, and Baishakhi Ray. 2024. Code-aware prompting: A study of coverage guided test generation in regression setting using LLM. *arXiv preprint arXiv:2402.00097* (2024).

[26] Lin Shi, Fangwen Mu, Yumin Zhang, Ye Yang, Junjie Chen, Xiao Chen, Hanzhi Jiang, Ziyou Jiang, and Qing Wang. 2022. Buglistener: identifying and synthesizing bug reports from collaborative live chats. In *Proceedings of the 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. IEEE, 299–311.

[27] Forrest Shull, Janice Singer, and Dag IK Sjøberg. 2007. *Guide to advanced empirical software engineering*. Springer.

[28] Yanqi Su, Dianshu Liao, Zhenchang Xing, Qing Huang, Mulong Xie, Qinghua Lu, and Xiwei Xu. 2024. Enhancing exploratory testing by large language model and knowledge graph. In *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*. IEEE, 1–12.

[29] Liangtai Sun, Yang Han, Zihan Zhao, Da Ma, Zhennan Shen, Baocai Chen, Lu Chen, and Kai Yu. 2024. Scieval: A multi-level large language model evaluation benchmark for scientific research. In *Proceedings of 2024 the 38th AAAI Conference on Artificial Intelligence (AAAI)*, Vol. 38. 19053–19061.

[30] Xiaoxuan Wang, Ziniu Hu, Pan Lu, Yanqiao Zhu, Jieyu Zhang, Satyen Subramaniam, Arjun R Loomba, Shichang Zhang, Yizhou Sun, and Wei Wang. 2023. Scibench: Evaluating college-level scientific problem-solving abilities of large language models. *arXiv preprint arXiv:2307.10635* (2023).

[31] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. 2022. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682* (2022).

[32] Chunqiu Steven Xia, Yinlin Deng, and Lingming Zhang. 2024. Top leaderboard ranking= top coding proficiency, always? EvoEval: Evolving coding benchmarks via LLM. *arXiv preprint arXiv:2403.19114* (2024).

[33] Chunqiu Steven Xia, Yuxiang Wei, and Lingming Zhang. 2023. Automated program repair in the era of large pre-trained language models. In *Proceedings of the 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1482–1494.

[34] Liang Xu, Anqi Li, Lei Zhu, Hang Xue, Changtai Zhu, Kangkang Zhao, Haonan He, Xuanwei Zhang, Qiyue Kang, and Zhenzhong Lan. 2023. Superclue: A comprehensive chinese large language model benchmark. *arXiv preprint arXiv:2307.15020* (2023).

[35] Jialu Zhang, José Pablo Cambronero, Sumit Gulwani, Vu Le, Ruzica Piskac, Gustavo Soares, and Gust Verbruggen. 2024. PyDex: Repairing bugs in introductory python assignments using LLMs. *Proceedings of the ACM on Programming Languages* 8, OOPSLA1 (2024), 1100–1124.

[36] ZhiPuAI. 2024. ChatGLM. https://en.chatglm.cn/. (Accessed on 05/18/2024).

[37] Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. 2023. Agieval: A human-centric benchmark for evaluating foundation models. *arXiv preprint arXiv:2304.06364* (2023).