Evaluating Fault Localization and Program Repair Capabilities of Existing Closed-Source General-Purpose LLMs

Shengbei Jiang†
Beijing Jiaotong University
Beijing, China
21291232@bjtu.edu.cn

Jiabao Zhang†
Beijing Jiaotong University
Beijing, China
21281296@bjtu.edu.cn

Wei Chen†
Beijing Jiaotong University
Beijing, China
21281275@bjtu.edu.cn

Bo Wang‡
Beijing Jiaotong University
Beijing, China
wangbo_cs@bjtu.edu.cn

Jianyi Zhou
Huawei Cloud Computing Technologies Co., Ltd.
Beijing, China
zhoujianyi2@huawei.com

Jie M. Zhang
King’s College London
London, United Kingdom
jie.zhang@kcl.ac.uk

ABSTRACT
Automated debugging is an emerging research field that aims to automatically find and repair bugs. In this field, Fault Localization (FL) and Automated Program Repair (APR) gain the most research efforts. Most recently, researchers have adopted pre-trained Large Language Models (LLMs) to facilitate FL and APR and their results are promising. However, the LLMs they used either vanished (such as Codex) or outdated (such as early versions of GPT). In this paper, we evaluate the performance of recent commercial closed-source general-purpose LLMs on FL and APR, i.e., ChatGPT 3.5, ERNIE Bot 3.5, and IFlytek Spark 2.0. We select three popular LLMs and evaluate them on 120 real-world Java bugs from the benchmark Defects4J. For FL and APR, we designed three kinds of prompts for each, considering different kinds of information. The results show that these LLMs could successfully locate 53.3% and correctly fix 12.5% of these bugs.

CCS CONCEPTS
 Software and its engineering → Search-based software engineering; Software testing and debugging.

KEYWORDS
Large Language Model, Fault Localization, Program Repair, Software Debugging

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

1 INTRODUCTION
Bugs are ubiquitous in modern software systems, threatening our daily lives. However, program debugging is time-consuming and challenging, consuming more than half of the developers’ programming time [5]. Hence, a large body of research efforts have been dedicated to automated debugging techniques.

Among these debugging techniques, Fault Localization (FL) techniques and Automated Program Repair (APR) approaches are emerging fields and gaining traction. FL techniques aim to automatically localize buggy code elements (e.g., lines or methods) [9–11, 17], while APR techniques aim to automatically generate patches to fix buggy programs without human intervention [18–20, 23]. FL and APR techniques have adopted deep-learning models to facilitate the capture and comprehension of the context of the buggy code [10, 11, 17, 22], and deep-learning-based approaches have been recognized as the state of the art. However, their performance is still limited because their training data only contains buggy code extracted from development history.

Most recently, with the overwhelming success of Large Language Models (LLM), researchers have proposed to harness LLMs for FL and APR [3, 4, 7, 14, 16]. Currently, with ChatGPT becoming more and more popular, researchers in the field of debugging predominantly utilize various versions of the ChatGPT models. For example, Fan et al. employed the latest model of Codex in 2022, while Xia et al. employed the latest model of ChatGPT in 2023, (i.e., gpt-3.5-turbo) [16]. However, with the emergence of more and more general-purpose LLMs, the capabilities of other models in debugging have yet to be evaluated.

In this paper, we aim to evaluate the capabilities of existing closed-source general-purpose LLM in FL and APR. More specifically, we performed a preliminary case study on 120 real-world Java bugs from six projects of the benchmark Defects4J [6], by comparing three commercial closed-source LLMs, namely ChatGPT 3.5, ERNIE Bot 3.5, and IFlytek Spark 2.0. We designed three types of prompts by emphasizing different kinds of information, including source code context, failure traces, and test code context. In contrast to existing work, we have employed several other closed-source
LLMs apart from ChatGPT and considered transforming the stack traces of failure traces into prompts.

The evaluation results show that all three LLMs successfully locate more than half of all bugs (i.e., 64/120) and correctly fix 12.5% (15/120). ChatGPT 3.5 achieves the best performance and IFlytek Spark archives the second best. Contrary to our expectation that providing more information would enhance the performance of LLMs, surprisingly, the prompts that only code the source code of the buggy method achieve the best FL performance. The prompts that copy the buggy statement achieve the best APR performance.

2 STUDY DESIGN

Figure 1 illustrates the overview of our study. Given a bug from the Defects4J benchmark, we generate prompts by tailoring the predefined prompt template and the bug context information extracted from code and test failure messages. We designed different prompt templates for FL and APR, respectively. We send the prompts to the target LLMs and get their results. Note that if the length of the prompt exceeds the token limit of the LLM (e.g., ChatGPT 3.5 has a 4096 token limit), we truncate it. Then we manually compare the results with the developer patches of Defects4J, which are used as the oracle. Note that despite ChatGPT, few LLMs provide an API interface, which makes us engage in conversation with these models through their web pages.

All experiments were conducted on a Legion Y9000P PC, with 16GB memory and an Intel i9-13900H CPU.

2.1 Research Questions

In this study, we aim to investigate the following research questions:

- RQ1: How do different LLMs perform for FL?
- RQ2: How do different LLMs perform for APR?
- RQ3: Do different sources of information help with localization and repair?

2.2 Chosen Bugs

We directly the popular benchmark Defects4J v1.2.0 [6], which contains 395 real-world Java bugs from six open-source projects. Defects4J is considered to be the baseline benchmark in the field of FL [1, 8, 10, 11, 21] and APR [18–20, 23].

Note that some bugs of Defects4J may cover multiple methods or even multiple files, resulting in an inability to measure FL and APR performance in a single conversation with LLMs. So we skip these bugs and only keep the bugs whose patch only affects a single method. As a preliminary study, we pick the first 20 bugs of each project, and thus in total, we perform our study on 120 bugs.

2.3 Chosen Models

We use the following criteria for selecting LLMs in this study.

1. The LLMs should be available to external users.
2. The LLMs should support coding missions.
3. The LLMs should be closed-source.

Currently, research on the performance of LLMs in software engineering tasks mainly focuses on open-source ones (e.g., CodeLlama) and ChatGPT. However, many novel commercial LLMs are continuously being proposed, and their code capabilities also need to be evaluated. At last, we select three LLMs, namely ChatGPT 3.5, ERNIE Bot 3.5, and IFlytek Spark 2.0. All these models are generative rather than infilling, but the parameter sizes are not officially released.

2.4 Fault Localization Experiment Settings

2.4.1 Prompt design. In our study, we directly use a zero-shot fashion single conversation to perform FL. To be more specific, we employ three types of information, namely source method code (i.e., src), bug-trigger assertion code (i.e., assert), and stack trace (i.e., stack). We combine them with different prompt templates as follows.

- FL Prompt 1: src + “There is a bug in the above code, please help me locate it.”
- FL Prompt 2: src + stack + “There is a bug in the above code, please help me locate it by considering the stack trace.”
- FL Prompt 3: src + stack + assert + “There is a bug in the above code, please help me locate it by considering the stack trace information and failure assertion code.”

2.4.2 Evaluation metric. As mentioned before, the bugs used in our study are limited within a single method. Thus we judge the line returned by LLMs by checking it is in the lines that are affected by the corresponding developer patch, which is also a common practice in the FL community. For the bugs that have multiple hunks, we consider a bug to be successfully located if all the hunks are reported as buggy.

2.5 Automated-Program Repair Experiment Settings

2.5.1 Prompt design. In the APR experiment, we also use a zero-shot fashion single conversation. Different from our FL experiment, we only use src information. Additionally, to meet the generative task, we kept the code before the bug and the bug itself for LLM to fix. In our pilot study, we found that LLMs rarely correctly fix the bugs without perfect localization, so we give either the line number of the bug or directly use the buggy statement. We combine them with different prompt templates as follows, where X is the placeholder of the line number of the bug and S is the buggy statement literal.

- APR Prompt 1: src + “There is a bug in line X of the code, please help me fix it.”
- APR Prompt 2: src + “There is a bug in S, please help me fix it.”
- APR Prompt 3: src + “There is a bug in the last statement, please help me fix it.”

2.5.2 Evaluation metric. Following existing APR common practice, we manually compare patches made by developers. If they match entirely, we consider them correct. If they don’t match, we further assess whether they are semantically equivalent. Note that manually checking the semantic equivalence is a common practice in the field of APR. If the patches are semantically equal to the developers’ patch, we mark it as correct, otherwise, we mark it as incorrect.
3 EVALUATION RESULTS

3.1 RQ1: FL Results
We merge all the results of three kinds of prompts, and Table 1 illustrates the FL performance of the LLMs. ChatGPT 3.5, which achieves the best performance, successfully locates 47 (i.e., about 39%) bugs with a single conversation. Figure 2 shows the intersection of successfully localized bugs of the three LLMs. In total, the three LLMs locate 64 out of the 120 bugs (about 53%). We can find that all three LLMs have a significant number of uniquely localized bugs, and ChatGPT 3.5 still has the most uniquely located bugs by locating 25 bugs that other LLMs could not.

![Figure 2: The intersection of successfully localized bugs of chosen LLMs.](image)

<table>
<thead>
<tr>
<th>Project</th>
<th>ChatGPT 3.5</th>
<th>ERNIE Bot 3.5</th>
<th>IFlytek Spark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chart</td>
<td>11</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>Lang</td>
<td>7</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Math</td>
<td>9</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>Time</td>
<td>6</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Mockito</td>
<td>7</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>Closure</td>
<td>7</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>47</td>
<td>25</td>
<td>25</td>
</tr>
</tbody>
</table>

![Table 1: The number of successfully localized bugs.](image)

3.2 RQ2: APR Results
Following the settings of the FL experiments, we merge all the repair results of three kinds of prompts. Table 2 shows the results, indicating that different from the performance of FL, there is no significant difference in the APR performance. All three LLMs only successfully fix less than 10% of the 120 bugs, and ChatGPT 3.5 achieves the best. Figure 3 shows the intersection of successfully fixed bugs of the three LLMs. In total, all three LLMs successfully fixed 15 out of the 120 bugs (i.e., 12.5%). Different from the Venn diagram of FL, there is no significant difference concerning the number of uniquely fixed bugs. In addition, the portion of the bugs that are fixed by all LLMs is large.

![Figure 3: The intersection of successfully repaired bugs of chosen LLMs.](image)

<table>
<thead>
<tr>
<th>Project</th>
<th>ChatGPT 3.5</th>
<th>ERNIE Bot 3.5</th>
<th>IFlytek Spark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chart</td>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Lang</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Math</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Time</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Mockito</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Closure</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Total</td>
<td>9</td>
<td>6</td>
<td>8</td>
</tr>
</tbody>
</table>

![Table 2: The number of successfully repaired bugs.](image)
We sincerely thank the anonymous reviewers for their valuable comments on this study. This work is supported by the National Natural Science Foundation of China under Grant No. 62202040.

REFERENCES