Towards Automatic Software Faults Analysis

Jiongchi Yu*
jcyu.2022@phdcs.smu.edu.sg
Singapore Management University
Singapore

Weipeng Jiang*
lenijwp@stu.xjtu.edu.cn
Xi'an Jiaotong University
China

Xiaoyu Zhang†
joshiningrain@gmail.com
Nanyang Technological University
Singapore

Qiang Hu†
qianghu@tju.edu.cn
Tianjin University
China

Xiaofei Xie
xfxie@smu.edu.sg
Singapore Management University
Singapore

Chao Shen
chaoshen@xjtu.edu.cn
Xi'an Jiaotong University
China

Abstract

Understanding software faults is fundamental to empirical research on software development and maintenance. Traditional fault analysis, although effective, typically relies on multiple human expert-driven steps, including software fault information collection, filtering, manual investigation, and classification. These processes are labor-intensive and time-consuming, creating significant bottlenecks for large-scale fault studies in complex and critical software systems with abundant faults and limiting iterative empirical research amid rapid software evolution.

In this paper, we decompose empirical software fault studies into three key phases and conduct an exploratory study on applying large language models (LLMs) to support fault analysis in open-source software. Our results show that LLMs can substantially improve analysis efficiency, reducing processing time to approximately two hours compared to weeks of manual effort. While LLMs demonstrate promising performance in identifying bug-related issues, they remain limited in accurately classifying fault categories. These findings highlight both the potential and current challenges of using LLMs to automate empirical software fault studies.

CCS Concepts

• **Software and its engineering** → **Software defect analysis**.

Keywords

Empirical Study, Large Language Model, Automated Research

ACM Reference Format:
Jiongchi Yu, Weipeng Jiang, Xiaoyu Zhang, Qiang Hu, Xiaofei Xie, and Chao Shen. 2026. Towards Automatic Software Faults Analysis. In *2026 IEEE/ACM 48th International Conference on Software Engineering (ICSE-Companion '26), April 12–18, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3774748.3795652

*Both authors contributed equally to this research.
†Corresponding authors.

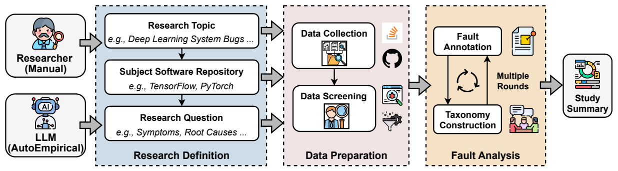


Figure 1: Workflow of empirical studies on software faults.

1 Introduction

Empirical software fault studies are critical for research and practice but remain highly labor-intensive, relying on manual data collection and annotation by experts. As modern software systems generate large volumes of issues under frequent updates, this manual process creates a major scalability bottleneck, limiting the timeliness and impact of fault analysis [4].

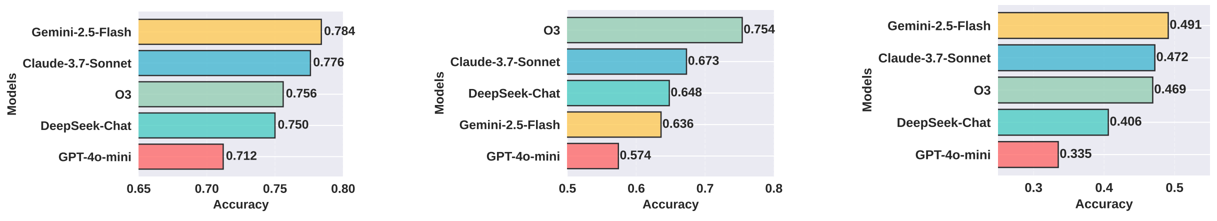
Recent advances in large language models (LLMs) offer new opportunities to address this challenge. Prior studies show that LLMs can effectively understand natural language and source code, enabling the automation of research tasks traditionally performed by human experts [2, 7, 11]. These capabilities have given rise to *auto-research*, where LLM-powered frameworks automate domain-specific research workflows [8]. Empirical software fault studies are particularly suitable for such automation, as they follow structured workflows and emphasize consistent judgment rather than novel algorithm design, aligning well with the strengths of LLMs [12].

Motivated by this observation, we propose an exploratory pipeline that integrates LLMs into core stages of empirical software fault studies. The pipeline decomposes the process into three phases: *research definition*, *data preparation*, and *fault analysis*. We evaluate LLM outputs against expert-established ground truth from prior studies [3, 6, 9, 10]. Using a representative empirical study, we show that the LLM-driven pipeline completes in approximately two hours, achieving a 20× speed-up over manual analysis. While LLMs demonstrate strong potential in identifying fault-related issues, their accuracy in fine-grained fault classification remains limited, highlighting both their promise and current limitations. To support reproducibility, we release our code on the project website [1].

2 Automated Software Fault Analysis

As shown in Figure 1, we decompose empirical software fault analysis into three stages and contrast human-driven practices with LLM-enabled alternatives, illustrating how it can be automated.

Stage I: Research Definition. Usually, human experts define the research scope, select representative software systems, and



(a) Bug-Related Issue Filtering **(b) Bug Symptoms Classification** **(c) Bug Root Causes Classification**

Figure 2: Overall performance of different LLMs across the three stages.

formulate research questions based on domain knowledge, which requires familiarity with the target ecosystem and common fault types. In our design, this expert-driven process is augmented by LLMs: given high-level research goals, LLMs help identify relevant repositories and suggest candidate research questions, supporting conceptual and exploratory tasks.

Stage II: Fault-Related Issue Selection. Conventionally, researchers manually screen large collections of issues or commits to identify fault-related cases by applying predefined inclusion and exclusion criteria. This process is highly labor-intensive and depends on expert judgment to distinguish true fault reports from non-fault issues. In our pipeline, candidate issues are first collected automatically, while the screening task is delegated to LLMs. Guided by expert-defined criteria, LLMs analyze issue metadata to identify fault-related cases, replacing manual inspection while remaining consistent with established empirical methodologies.

Stage III: Fault Taxonomy Classification. Traditionally, fault taxonomy classification requires experts to interpret issue descriptions and assign fault symptoms and root causes based on predefined taxonomies, demanding substantial domain expertise. In this study, we adopt fixed taxonomic frameworks from prior literature to preserve expert knowledge. LLMs are then used to map fault-related issues to corresponding taxonomy categories, automating expert annotation while maintaining alignment with established classifications. We leave the construction and refinement of taxonomies through multi-agent LLM reasoning as future work.

3 Preliminary Experiment

Experiment Setup. We evaluate the proposed pipeline using an established empirical fault study [5, 9]. Following the original study, we construct a balanced subset of 500 issues by sampling 250 fault-related and 250 non-fault cases from 3,859 candidates. For Stage III, we use the full set of 684 issues with expert-labeled fault symptoms and root causes as ground truth. We evaluate five state-of-the-art LLMs (Gemini-2.5-Flash, Claude-3.7-Sonnet, OpenAI o3, DeepSeek-Chat, GPT-4o-mini) and report accuracy as the primary metric to compare LLM outputs against expert annotations.

Performance Across Stages. As shown in Figure 2, LLMs exhibit distinct performance characteristics across the three stages. In Stage I, they can identify well-known representative projects and formulate reasonable research questions, but tend to focus on prominent repositories and overlook smaller or third-party projects. In Stage II, all evaluated LLMs achieve promising results in identifying fault-related issues, substantially improving efficiency over manual screening despite some false positives. In contrast, Stage III remains challenging. Even with predefined taxonomies, LLMs struggle with fine-grained fault classification, achieving at most around 50% accuracy for root cause identification. Our results suggest that while LLMs are effective at automating early-stage screening, precise fault categorization still requires deeper semantic understanding.

4 Future Plans

Our preliminary results highlight both the potential and current limitations of LLM-driven automation for empirical fault analysis. We plan to extend this work in two directions.

Comprehensive Benchmark Construction. We plan to build a comprehensive benchmark for automated empirical fault studies by aggregating and replicating datasets from recent top-tier software engineering research. This benchmark will provide standardized, expert-annotated data to support reproducible evaluation.

Multi-Agent Systems for Automated Empirical Studies. We aim to develop a multi-agent LLM system in which specialized agents collaboratively perform fault screening, taxonomy-based classification, and iterative analysis refinement, enabling more scalable and end-to-end automated empirical studies.

References

[1] AutoEmpirical. 2025. Website. https://sites.google.com/view/autoempirical.
[2] Sébastien Bubeck et al. 2023. Sparks of artificial general intelligence: early experiments with GPT-4 (2023). *arXiv:2303.12712* 1 (2023).
[3] Zhenpeng Chen et al. 2021. An empirical study on deployment faults of deep learning based mobile applications. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 674–685.
[4] Steffen Herbold et al. 2022. Problems with SZZ and features: An empirical study of the state of practice of defect prediction data collection. *Empirical Software Engineering* 27, 2 (2022), 42.
[5] Qiang Hu et al. 2023. Towards understanding model quantization for reliable deep neural network deployment. In *2nd IEEE/ACM International Conference on AI Engineering-Software Engineering for AI, CAIN 2023*.
[6] Weipeng Jiang et al. 2025. The Foundation Cracks: A Comprehensive Study on Bugs and Testing Practices in LLM Libraries. *arXiv:2506.12320* (2025).
[7] Qusai Khraisha et al. 2023. Can large language models replace humans in the systematic review process? Evaluating GPT-4's efficacy in screening and extracting data from peer-reviewed and grey literature in multiple languages. *arXiv:2310.17526* (2023).
[8] Chengwei Liu et al. 2025. A vision for auto research with llm agents. *arXiv:2504.18765* (2025).
[9] Lili Quan et al. 2022. Towards understanding the faults of javascript-based deep learning systems. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–13.
[10] Jiongchi Yu et al. 2024. Bugs in pods: Understanding bugs in container runtime systems. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 1364–1376.
[11] Jiongchi Yu et al. 2025. Chimera: Harnessing multi-agent llms for automatic insider threat simulation. *arXiv:2508.07745* (2025).
[12] Lianmin Zheng et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems* 36 (2023), 46595–46623.