

Demystifying Issues, Causes and Solutions in LLM Open-Source Projects

Yangxiao Cai^a, Peng Liang^{a,*}, Yifei Wang^a, Zengyang Li^b and Mojtaba Shahin^c

^aSchool of Computer Science, Wuhan University, China

^bSchool of Computer Science & Hubei Provincial Key Laboratory of Artificial Intelligence and Smart Learning, Central China Normal University, China

^cSchool of Computing Technologies, RMIT University, Australia

ARTICLE INFO

Keywords:

Large Language Model, GitHub, Open Source Project, Issue, Cause, Solution

ABSTRACT

With the advancements of Large Language Models (LLMs), an increasing number of open-source software projects are using LLMs as their core functional component. Although research and practice on LLMs are capturing considerable interest, no dedicated studies explored the challenges faced by practitioners of LLM open-source projects, the causes of these challenges, and potential solutions. To fill this research gap, we conducted an empirical study to understand the issues that practitioners encounter when developing and using LLM open-source software, the possible causes of these issues, and potential solutions. We collected all closed issues from 15 LLM open-source projects and labelled issues that met our requirements. We then randomly selected 994 issues from the labelled issues as the sample for data extraction and analysis to understand the prevalent issues, their underlying causes, and potential solutions. Our study results show that (1) *Model Issue* is the most common issue faced by practitioners, (2) *Model Problem*, *Configuration and Connection Problem*, and *Feature and Method Problem* are identified as the most frequent causes of the issues, and (3) *Optimize Model* is the predominant solution to the issues. Based on the study results, we provide implications for practitioners and researchers of LLM open-source projects.

1. Introduction

With the advancements of Pre-trained Language Models (PLMs) in recent years, there has been a significant breakthrough in the capacity of Language Models (LMs) to handle very large-scale data, which has led to the emergence of Large Language Models (LLMs) (Hou et al., 2024; Zhao et al., 2023a). LLMs refer to neural network language models trained on massive text datasets with billions of parameters. The most advanced LLMs to date have demonstrated remarkable language comprehension and generation capabilities (Zheng et al., 2023).

Since the release of ChatGPT in 2022 (OpenAI, 2022), there has been a noticeable increase in research related to LLM (Zhao et al., 2023a). The Software Engineering (SE) research community has also extensively used LLMs as tools to solve SE tasks. This is evident in recent review papers on LLM in SE. For example, Zheng *et al.* collected relevant literature on LLMs from seven literature databases, categorizing these papers according to the SE tasks involved, then reviewed the current research status of LLMs from the perspective of the seven major tasks in software development (Zheng et al., 2023). In addition, Hou *et al.* conducted a systematic literature review on the application of LLMs in SE (Hou et al., 2024). They investigated which LLMs were utilized for SE tasks, and the collection and preprocessing of SE-related datasets for these models. They also investigated

strategies for optimizing and evaluating LLM performance in SE, and the SE tasks LLMs successfully addressed.

Conversely, software practitioners are increasingly integrating LLMs as components within their software systems (Weber, 2024; Liu et al., 2023). These systems, known as “LLM-based Systems”, utilize the capabilities of LLMs to perform tasks that typically require substantial coding effort (Weber, 2024). This trend is also evident in open-source software (OSS) projects, primarily due to the emergence of open-source LLMs and frameworks. For example, LangChain, an open-source software framework for building applications based on LLMs, is currently receiving significant attention from practitioners (Chase, 2024). In addition, Microsoft’s semantic-kernel, an SDK that integrates LLMs into popular programming languages such as C#, Python, and Java, has also received widespread attention (Microsoft, 2024). In this paper, we refer to OSS projects that leverage LLMs as **LLM open-source projects**.

Despite the increasing number of LLM open-source projects, there is no systematic research on the issues experienced in LLM open-source project development, the causes of the issues, and potential solutions from the perspective of practitioners. This study aims to address this gap by studying approximately 1000 GitHub closed issue discussions from LLM open-source projects. Considering that GitHub is presently the largest hosting platform for open-source projects in the world, and numerous related studies have utilized community data from open-source software for empirical software engineering work, we have decided to select LLM open-source projects from GitHub.

Our findings show that: (1) *Model Issue* is the most common issue faced by practitioners, (2) *Model Problem*, *Configuration and Connection Problem*, and *Feature and*

*Corresponding author.

✉ yangxiaocai@whu.edu.cn (Y. Cai); liangp@whu.edu.cn (P. Liang);

whiten@whu.edu.cn (Y. Wang); zengyangli@ccnu.edu.cn (Z. Li);

mojtaba.shahin@rmit.edu.au (M. Shahin)

ORCID(s):

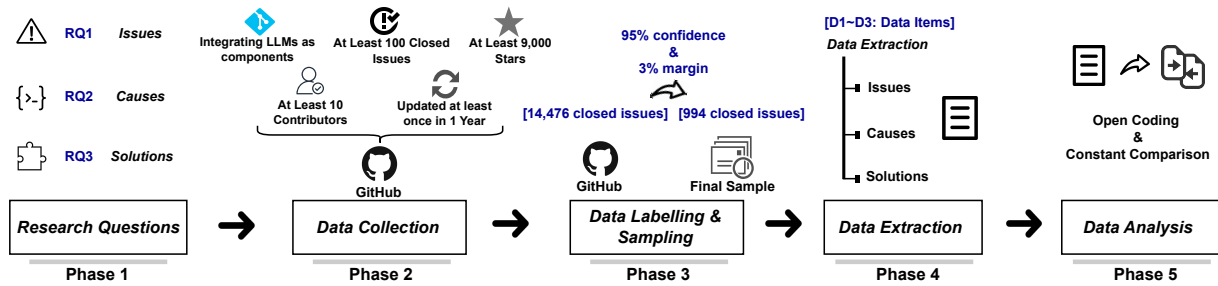


Figure 1: Overview of the research process

Method Problem are identified as the most frequent causes of the issues, and (3) *Optimize Model* is the predominant solution to the issues.

The **main contributions** of this work: (1) We identified the issues occurred in the development and utilization of LLM open-source software with the dataset (Cai et al., 2024) and provided a two-level taxonomy for these issues. (2) We identified the causes and solutions of these issues, and came up with a taxonomy for these causes and solutions. (3) We provided the mapping relationship between the identified issues and causes, as well as the mapping relationship between the issues and solutions.

The rest of this paper is structured as follows: Section 2 presents the Research Questions (RQs) and the research process employed in this study. Section 3 presents the study results with their interpretation. Section 4 discusses the implications based on the study results. Section 5 outlines the potential threats to validity. Section 6 reviews the related work, and Section 7 concludes this work along with the directions for future research.

2. Methodology

Our study aims to identify the issues in developing and using LLM open-source software and to determine the causes of these issues as well as possible solutions. With this aim in mind, we formulate three RQs to guide the subsequent phases of the methodology, as shown in Fig. 1. The RQs and their rationales are detailed in Section 2.1.

2.1. Research Questions

RQ1: What issues do OSS practitioners encounter when developing and using LLM open-source software?

Rationale: As mentioned in Section 1, remarkable advances in LLMs (Zhao et al., 2023a) have led to a growing number of open-source projects leveraging LLMs as one of their core functional components (i.e., LLM open-source projects). The answer of this RQ can provide an overall picture of the issues that OSS developers experience when developing LLM open-source projects. It also opens new directions for subsequent research on how these issues can be resolved.

RQ2: What are the underlying causes of these issues?

Rationale: After collecting the issues currently faced by practitioners as identified in RQ1, it is important to further

identify the causes of these issues. By identifying the causes, the study can provide insights for finding solutions and inspire subsequent refinement and optimization of both LLM open-source software and the LLMs.

RQ3: What are the current and potential solutions to address these issues?

Rationale: This RQ aims to understand how practitioners address the issues they encounter. The answer to this RQ can identify the most commonly used solutions, and provide various potential solutions from practitioners to the issues collected in RQ1. Ultimately, by exploring existing solutions to these issues, the study can provide guidance for the optimization of the software, thereby enhancing its performance.

2.2. Data Collection

We collected the data for our study from GitHub, which is currently the largest platform for hosting open-source software projects. In addition, every project on GitHub has an “Issues” page, which provides a platform for developers and users of the project to raise issues encountered while developing and using the software, and to discuss the identification, tracing, and resolution of these issues with other developers and users. In order to thoroughly explore the causes of issues and their potential solutions, we decided to collect all “closed” issues from the selected LLM open-source project repositories.

To collect data as comprehensively as possible, we did not set a start date for the source of the data. The date we conducted the data search is December 11, 2023, and we collected the data that were created before this date. The first author utilized a keyword search approach, connecting the search terms “LLM” and “Large Language Model” with an “OR” logic, and then searched for projects on the entire GitHub. Any project that contains the search terms (e.g., project description, tags) was retrieved.

After obtaining the search results, we found that although many projects include “LLM” or “Large Language Model” terms, they do not use LLMs as components to implement their functionality. For example, some project are to teach LLMs without using LLMs in the projects. Therefore, we set the following criteria for selecting LLM open-source projects:

Table 1

Information of selected LLM open-source projects

Project Name	#Issues	#Contributors	#Forks	#Stars
langchain	3475	1991	10.6k	71.9k
gpt4all	901	79	6.1k	57k
MetaGPT	97	45	3.7k	32.5k
FastChat	919	206	3.7k	30.3k
text-generation-webui	2561	264	3.9k	29.8k
ollama	552	89	1.1k	27.6k
quivr	875	95	3k	26.4k
autogen	223	359	2.2k	18.8k
ChatDev	149	40	2.1k	18.5k
unilm	745	54	2.2k	16.7k
semantic-kernel	1030	195	2.2k	15.6k
SuperAGI	269	61	1.6k	13.4k
haystack	2381	232	1.5k	12k
dolly	157	13	1.2k	10.0k
pandas-ai	254	59	782	9.3k

- (1) The project must be a real open-source software, which indicates that the project must be a real open-source software project that integrates LLMs as components (e.g., APIs) to implement its main functionality. They should not be a course teaching project, teaching guide project, LLM tutorial, LLM-related books, examination or competition related to LLMs, or other projects that merely use our keyword in project documentation without employing an LLM framework or functionality.
- (2) The project must have at least 100 closed issues.
- (3) The project must have at least 9,000 stars.
- (4) The project must have at least 10 contributors.
- (5) The project must have been updated at least once within the past year.

In our criteria, we defined criterion (1) to filter projects that utilize LLMs as components to implement their main functionality Malavolta et al. (2021). Similar to other empirical studies (e.g., Waseem et al. (2023); Zheng et al. (2024)) that employed project information (e.g., star count) as criteria for selecting appropriate GitHub projects with sufficient popularity and representativeness, we defined criteria (2), (3), (4), and (5) to exclude unpopular, inactive, or non-representative LLM open-source projects.

Following the selection criteria, the first author ultimately selected 15 LLM open-source projects (see Table 1) that met our criteria. The domains of the 15 selected LLM open-source projects are categorized into five categories (see Table 2). Then, the first author employed web scrapers to collect all the closed issues from these project repositories. In the end, we gathered a total of 14,588 closed issues. Each closed issue was recorded in an MS Excel file. We named each MS Excel file using the convention: 'PROJECT NAME' + '_' + 'ISSUE NUMBER'.

2.3. Data Labelling and Sampling

After collecting closed issues from the 15 LLM open-source projects, we found that some issues raised by practitioners may not be related to the problems with the projects.

Table 2

Domain categorization of the selected LLM open-source projects

Domain	#Definition	#Projects
LLM Development Framework	Open-source projects that provide frameworks for developing software by leveraging LLMs	langchain, MetaGPT, quivr, autogen, ChatDev, semantic-kernel, SuperAGI, haystack, dolly
LLM Running Tool	Open-source projects for running LLMs in specific environments	gpt4all, FastChat, ollama
LLM Data Analysis Tool	Open-source projects for data analysis using LLMs	pandas-ai
LLM Training Tool	Open-source projects that support pre-training across tasks, languages, and modalities	unilm
LLM Frontend Tool	Open-source projects that develop user interface for LLMs	text-generation-webui

For example, some practitioners only raised an issue to chat with others or to learn the features of GitHub, like a practitioner simply submitted an issue titled “hi” as a greeting, without any content related to the project. In addition, not all issues have corresponding category labels (e.g., pandas-ai issue #85). Relying solely on existing labels could lead to the omission of critical information. Therefore, we conducted data labelling to identify issues suitable for our study. The closed issues we expected should contain specific information related to the issues that practitioners encountered while developing or using LLM open-source software. For example, a user mentioned in text-generation-webui #2926 that “the model is loaded into memory without any errors, but crashes on generation of text”, which explicitly indicates an issue present in the LLM open-source project text-generation-webui. Then, on the basis of the pilot and formal labelling results (as detailed in Section 2.3.1 and Section 2.3.2, respectively), we selected a representative data sample (as detailed in Section 2.3.3) to construct the dataset for data extraction.

2.3.1. Pilot Data Labelling

To minimize potential personal biases, the first and third authors independently conducted a pilot data labelling. We randomly selected 50 closed issues from all closed issues, and then the first and third authors independently labelled them. The two authors labelled the data that contain information related to issues arising during the development or use of LLM open-source software. The inter-rater consistency on data labelling results between the two authors was measured using the Cohen's Kappa coefficient (Cohen, 1960), yielding a value of 0.838, which indicates a very high level of data labelling consistency between the two authors. For any disagreements or conflicts in the pilot data labelling results, the two authors discussed together with the second author to reach a consensus. Finally, we got 32 issues out of 50 issues are considered relevant. The results of pilot data labelling were compiled and recorded in MS Excel files (Cai et al., 2024).

2.3.2. Formal Data Labelling

The first author then conducted the formal data labelling. In this process, we excluded the issues not related to our study. The projects we selected utilized LLMs to achieve the main functions of the software. We only kept those issues related to the problems that practitioners encountered during the development and use of LLM open-source software. Ultimately, the first author collected a total of 14,476 closed issues. The results of formal data labelling were compiled and recorded in MS Excel files (Cai et al., 2024).

2.3.3. Data Sampling

We found that the dataset comprising a total of 14,476 closed issues was too large for manual data extraction. Therefore, we decided to select a suitable number of closed issues. We randomly selected 994 closed issues to form a representative set as our dataset for data extraction with a 95% confidence level and a 3% margin of error (Israel, 1992). The source projects of the closed issues in the final data sample are presented in Table 3. In addition, the distribution of the duration from issue opening to closing for the closed issues in the final data sample is shown in Fig. 2.

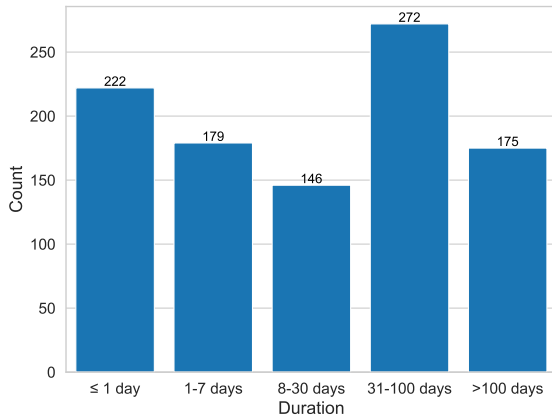


Figure 2: The distribution of the duration of the closed issues in the final data sample

2.4. Data Extraction

To answer the three RQs presented in Section 2.1, we defined a set of data items for data extraction, as shown in Table 4. The data items D1-D3 are respectively used to extract information about issues, underlying causes, and solutions that address these issues, corresponding to RQ1-RQ3. These three data items can be extracted from any part of the content of the collected closed issues, including the title, description, and discussion of the issue. To ensure the accuracy of the data extraction results and the validity of subsequent data analysis, we decided to conduct the data extraction manually. The other reason is that the data sample size was not very large, and manual data extraction was feasible. Therefore, the data extraction was conducted manually.

Table 3

The source projects of the closed issues in the final data sample

Project Name	#Selected Issues	#Contributors	#Lines of Code
langchain	226	1991	1,468.7k
gpt4all	50	79	238.1k
MetaGPT	9	45	133.1k
FastChat	63	206	121.7k
text-generation-webui	178	264	162.9k
ollama	49	89	276.4k
quivr	52	95	61.4k
autogen	19	359	1,037.4k
ChatDev	11	40	1,372k
unilm	50	54	2,047.9k
semantic-kernel	75	195	726.4k
SuperAGI	22	61	62.1k
haystack	162	232	140.3k
dolly	9	13	1.6k
pandas-ai	19	59	84.0k
Total	994		

2.4.1. Pilot Data Extraction

We randomly selected 50 closed issues from our dataset. Next, the first author independently conducted the pilot data extraction on the 50 issues. The results of pilot data extraction indicated that all three data items (see Table 4) could be extracted from our dataset. Based on the pilot data extraction results, we reached a consensus and defined the following rules for formal data extraction: (1) In principle, only one *issue* can be extracted from each closed issue. If there are multiple issues that can be extracted, we record the first issue discussed. (2) If there are multiple *causes* mentioned in a closed issue, we record all the causes of the issue we extract in the former step, which are identified by the reporter and the development team. (3) If there are multiple *solutions* mentioned in a closed issue, we record all the solutions identified by the reporter and the development team that can address the issue we extract in the former step.

2.4.2. Formal Data Extraction

After completing the pilot data extraction, the first author independently conducted the formal data extraction on the dataset to extract the data based on the data items defined in Table 4. If there were any questions during the data extraction process, the first author discussed these questions with the second author to resolve the doubts. After the first author completed the formal data extraction, the second author reviewed the extraction results and then discussed the results with the first author to reach a consensus on the inconsistencies. The first and second authors conducted multiple rounds of reviews and revisions on the data extraction results to obtain the final results. The data extraction results were compiled and recorded in an MS Excel file (Cai et al., 2024).

In the formal data extraction phase, we totally got 11 closed issues that discussed multiple issues in one closed issue in the final data sample. However, according to our criteria, we only kept the first issue discussed and its corresponding causes and solutions.

Besides, not all closed issues had a corresponding *cause* and *solution* which could be extracted. For some closed issues, the discussion content of the issues was too vague to discern specific information (i.e., *cause* or *solution*) that

could be extracted. Moreover, if an issue of a project is not discussed or updated for a long period, it will be automatically closed and turn into a closed issue, making it impossible to extract valid *cause* and *solution*. An issue may also be temporarily closed due to its low priority for resolution. Similarly, an issue might be too complex to identify its causes and solutions.

2.5. Data Analysis

After completing the data extraction, we conducted data analysis to answer the three RQs formulated in Section 2.1. We employed Open Coding and Constant Comparison methods for data analysis, which are commonly used methodologies within Grounded Theory for qualitative data analysis in software engineering research Stol et al. (2016). In Grounded Theory, Open Coding refers to the initial process of data analysis. In this process, researchers meticulously check the text to identify crucial concepts and categories, assigning descriptive codes to these elements. Constant Comparison is an approach used to refine and integrate these concepts and categories. In this process, researchers employ a bottom-up methodology to recurrently compare new data with previously coded data to refine and adjust existing categories based on their differences and similarities.

The steps in our data analysis are as follows: (1) The first author meticulously read through the text of each closed issue, then assigned descriptive codes that succinctly summarize the core themes according to the three RQs. For example, the issue of langchain #2174 was coded as “*Model Test Failure*”, which indicates that an error occurred while testing the model leading to failure. (2) The first author compared the commonalities and differences between codes. Following the principles of a bottom-up approach, the first author categorized identical or similar codes, and used consistent descriptive language to describe the higher-level types and categories. For example, the code from langchain #2174, together with other similar codes, constitutes the MODEL TEST ISSUE type, which further belongs to *Model Issue* category. (3) Once uncertainties about the code descriptions arose, the first author engaged in discussions with the second author to reach a consensus. Due to the nature of Constant Comparison, the initial results underwent several iterations and modifications before reaching the final results. (4) The second author reviewed and verified the initial analysis results. In case of any doubts or disagreements, the first and the second authors engaged in discussions to reach a final consensus and ultimately resolved the conflict. After resolving 31 conflicts in the categories of issues, 22 conflicts in the categories of causes, and 27 conflicts in the categories of solutions, we obtained the final results of the data analysis.

3. Results and Interpretation

In this section, we report the study results of the three RQs and provide their interpretation.

For the *Issue*, *Cause* and *Solution* taxonomies, we provide a two-tier classification, i.e., categories and types. We also provide brief descriptions of the types under each

Table 4

Data items extracted and their corresponding RQs

#	Data Item	Description	RQ
D1	Issue	<i>The most critical issue discussed in a GitHub issue. In principle, only one issue can be extracted from every closed issue.</i>	RQ1
D2	Cause	<i>The cause(s) for the issue is clearly clarified. Due to the complexity of some issues, there may have multiple causes.</i>	RQ2
D3	Solution	<i>The solutions(s) that can solve the issue is clearly provided. Some issues may have multiple solutions.</i>	RQ3

category. To show the relationship between issues and their causes and solutions, we present the mapping relationships between the *Issue* categories and the *Cause* categories, as well as between the *Issue* categories and the *Solution* categories. It should be noted that only closed issues from which a cause has been extracted are represented in the issue-cause mapping. Similarly, only closed issues from which a solution has been extracted are represented in the issue-solution mapping.

3.1. Category of Issues (RQ1)

Fig. 3 presents the taxonomy of the issues extracted from the dataset. It can be observed that *Model Issue* (24.55%) accounts for the majority of issues encountered by practitioners of LLM open-source projects. In addition, a substantial number of practitioners have raised *Component Issue* (9.26%) and *Parameter Issue* (9.26%). There is also a portion of practitioners who have encountered *Answer Issue* (8.25%), which indicates that they did not get answers they want from LLM open-source software, while smaller percentages (less than 8%) were identified for *Performance Issue* (7.14%), *Code Issue* (6.74%), *Installation Issue* (4.93%), *Documentation Issue* (4.73%), *Configuration Issue* (4.53%), *Network Issue* (4.43%), *Memory Issue* (3.92%), *Prompt Issue* (3.62%), *Security Issue* (3.52%), *GUI Issue* (2.72%), and *Database Issue* (2.41%).

3.1.1. Model Issue (24.55%)

Model Issue refers to issues related to the models that practitioners encounter while developing and using LLM open-source software. This category encompasses the following ten types.

- **MODEL RUNTIME ISSUE** refers to the issues that arise during the runtime of LLMs. The core functions of LLM open-source software are performed by the integrated models. Therefore, anomalies or crashes in the models during runtime have a significant impact on the operation of LLM open-source software. For example, a user found that “*the model is loaded into memory without any errors, but crashes on generation of text*” (text-generation-webui #2926).
- **MODEL ARCHITECTURE ISSUE** refers to issues related to the design and implementation of LLM architecture. Issues in this type are primarily manifested in unreasonable architectural design (e.g., layers) and unsuitable implementation of mechanisms (e.g., model

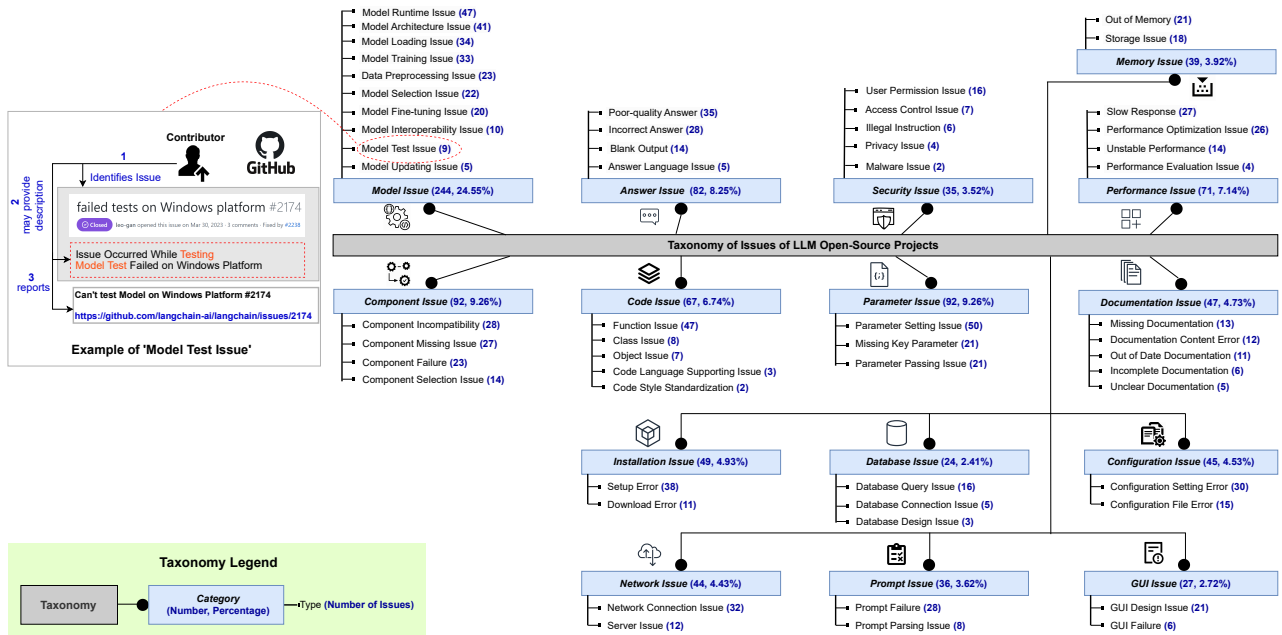


Figure 3: Taxonomy of issues of LLM open-source projects

training), leading to anomalies or failures in the training and use of LLMs. For example, a user received a message that “*InstructorEmbedding is not found*”, which is a component used to form the embedding layer of the model (langchain #867).

- **MODEL LOADING ISSUE** refers to the issues that arise during the loading of LLMs, such as loading failures, incomplete loading, loading the wrong model. Once the model fails to load, the subsequent functions of LLM open-source software cannot proceed normally. For example, a user reported that “*an error was thrown when loading with Exllama or Exllamav2 even though pip indicates they are installed*” (text-generation-webui #4293).
- **MODEL TRAINING ISSUE** refers to the issues that arise during the training of LLMs. These issues include inefficient training algorithms, the need for optimization of model training algorithms, sudden interruptions or crashes during training. For example, a user asked “*is DEiT-3 trained from scratch only based on the BEiT initialization algorithm or using the similar stagewise pre-training strategy introduced in VLMO*”, which indicates that doubts regarding the model training algorithm exist (unilm #1022).
- **DATA PREPROCESSING ISSUE** refers to the issues that arise during data preprocessing in LLMs, including issues in data segmentation, normalization, embedding, and other related processes. The quality of the data received by the models significantly affects the results of model training and operating. For example,

a user proposed to the development team that the text splitters could be enhanced to “*try and generate text fragments*”, which can aid users in better locating and citing specific portions of text (langchain #5451).

- **MODEL SELECTION ISSUE** refers to issues concerning choosing proper LLMs to implement required functionalities. Many LLM open-source projects support a variety of LLMs. Therefore, selecting one or multiple appropriate models to implement functions and accomplish tasks is crucial. For example, a user asked “*which trained model to choose for GPU-12GB, Ryzen 5500, 64GB*”, which indicates an issue of selecting models (gpt4all #324).
- **MODEL FINE-TUNING ISSUE** refers to the issues occurring when fine-tuning LLMs in order to adapt the models for different functions or software integration. To implement new features or incorporate a new model into the software, it is necessary to fine-tune the current models to complete the aforementioned tasks. However, fine-tuning LLMs may potentially lead to issues. For example, a user asked “*is there any example how to use it (or other scripts) to fine-tune this model*” (FastChat #1159).
- **MODEL COLLABORATION ISSUE** refers to the issues arising when multiple LLMs collaborate to complete tasks. When accomplishing certain tasks (e.g., information retrieval), it is necessary for several models to work together and interact with each other. However, issues such as incompatibility between models may

occur. For example, a user reported that “*default models should be published when Haystack 1.19.0 is released, to prevent incompatibility*” (haystack #5321).

- **MODEL TEST ISSUE** refers to the issues that arise during the testing of LLMs. When testing the models, issues may occur such as test failures, inconsistent or inaccurate test results, and incomplete or missing testing procedures. For example, a user “*got a timeout error*” when testing an LLM called GPT-JT (langchain #648).
- **MODEL UPDATING ISSUE** refers to the issues occurring during the model update process, which include update failures, incorrect update versions, update interruptions, and anomalies caused by updates. For example, a user reported that “*the new OpenAI client library released this week breaks the MT-bench evaluation script*”, which indicates that updating to the new version of the OpenAI client library caused the existing script to malfunction (FastChat #2657).

Interpretation: As a core functional component of LLM open-source software, practitioners are more concerned with functionality and performance of the model, making *Model Issue* be the most prevalent category of issues in developing and using LLM open-source software. **MODEL RUNTIME ISSUE** (47) and **MODEL ARCHITECTURE ISSUE** (41) are the top two types of *Model Issue*. **MODEL RUNTIME ISSUE** highlights the instability of LLMs at runtime and high demands for the deployment environment of LLM open-source software, while *Model Architecture Issue* indicates unreasonable connections between layers and unsuitable implementation of training mechanisms.

3.1.2. Component Issue (9.26%)

Component Issue refers to the issues that arise within or between the components of LLM open-source software. These issues may affect user experience and stability of LLM open-source software. This category encompasses the following four types.

- **COMPONENT INCOMPATIBILITY** refers to incompatibility issues between components of LLM open-source software, such as version incompatibility, incompatibility in data types, or even functional incompatibility. For example, a user reported that “*function calls do not work with Azure OpenAI currently due to differences between the OpenAI API and AzureOpenAI*” (autogen #78).
- **COMPONENT MISSING ISSUE** refers to the need for additional or extended components of LLM open-source software. The emergence of new requirements that necessitate a new component, or the realization that an essential function lacks a component for implementation. For example, to “*implement real-world, persistent Document Store for v2*”, a user suggested that the haystack team integrate a component called

“*ElasticSearch 8*”, which is an open-source search engine (haystack #5326).

- **COMPONENT FAILURE** refers to the malfunctions and crashes of components. Such failures constitute a significant issue affecting the functionality and the user experience of LLM open-source software. For example, a user reported that the API “*crashed on the second response*” when using a model named ggml-vicuna-7b (text-generation-webui #816).
- **COMPONENT SELECTION ISSUE** refers to the issues of choosing the right components in the development of LLM open-source software. When implementing the specific features, the selection of components is crucial. For example, a user requested the langchain team to support a “*multi-core loader*”, because the loader called *DirectoryLoader* was “*using a single core and missing the opportunity of leverage the multiple cores*” (langchain #3720).

Interpretation: *Component Issue* arises from the failure to use appropriate components to implement the corresponding functionality, or even the absence of required components. We found that under the *Component Issue* category, the top two types: **COMPONENT INCOMPATIBILITY** (28) and **COMPONENT MISSING ISSUE** (25), have similar proportions. It shows the poor compatibility between components developed by different developers and development communities. Additionally, older components may become incompatible with newer versions.

3.1.3. Parameter Issue (9.26%)

Parameter Issue refers to issues concerning the parameters of LLM open-source software. The definition, input, passing, and output of parameters are critical steps in the process of the development and use of LLM open-source software. The fact that *Parameter Issue* ranks third among all categories of issues emphasizes its importance. This category encompasses the following three types.

- **PARAMETER SETTING ISSUE** refers to the issues with the settings of parameters within LLM open-source software. Improper parameter settings may result in poor performance or even anomalies. The settings of parameters include not only the values of the parameters but also their formats, names, length constraints, and the definitions of their data structures. Ensuring reasonableness and consistency of parameter settings is crucial for facilitating subsequent passing and calculation. For example, a user suggested the langchain team that parameters, such as “*OPENAI_BASE_URL, OPENAI_API_KEY, CLAUDE2_BASE_URL, and CLAUDE2_API_KEY*”, can be replaced to increase flexibility in handling the integration and use of other APIs (ChatDev #6).
- **MISSING KEY PARAMETER** refers to the issues arising from the absence of certain parameters. There

is complex logic for parameter passing and calculation within LLMs. The lack of key parameters can significantly impact the training and effectiveness of the model. For example, a user reported that “*the parameter ‘sources’ is empty but it should be included in a list called ‘result’ as a string*” (langchain #5536).

- **PARAMETER PASSING ISSUE** refers to the issues that occur during the process of parameter passing in LLM open-source software. These issues manifest in aspects such as type conversion of parameters, the composition of parameter sets, and data formatting during the passing process. For example, a user was “*unable to provide ‘llm_chain’ to initialize_agent() while initializing the agent*” (langchain #4437).

Interpretation: *Parameter Issue* is a critical concern in the development and use of LLM open-source software. LLMs need to handle a large number of parameters, which presents significant challenges for developers. **PARAMETER SETTING ISSUE** (50) is the largest type of *Parameter Issue*, which indicates that there are issues with unreasonable parameter values, inappropriate parameter types, missing parameters, and redundant parameters in the development and use of LLM open-source software. Moreover, the complexity of large-scale parameters also presents a significant challenge for developers.

3.1.4. Answer Issue (8.25%)

Answer Issue refers to issues concerning the responses provided by LLM open-source software to the questions given by users. The effectiveness of the responses of LLM open-source software is a crucial characteristic that reflects the user experience of the software. This category encompasses the following four types.

- **POOR-QUALITY ANSWER** refers to responses produced by LLM open-source software that are correct but low quality, such as being overly verbose with excessive repetition, failing to generate responses according to the specified formats provided by users, and providing responses that are too brief and lack details. For example, when a user asked “*a few questions in a conversation, there is no context between questions*”, which indicates that the memory of the models is too short to give satisfactory answers (ollama #8).
- **INCORRECT ANSWER** refers to responses where wrong answers are provided by LLM open-source software, such as responses that are not pertinent to the question asked, factually inaccurate, or illogical. For example, a user reported that “*the results both from Azure OpenAI and from OpenAI are really random and have nothing to do with prompts*”, which indicates that answers are given randomly (semantic-kernel #1337).
- **BLANK OUTPUT** refers to issues that LLM open-source software fails to provide a response, resulting

in an empty answer, or is unable to successfully deliver a response. For example, a user reported that the answers given by the model “*are all blank*” (FastChat #2319).

- **ANSWER LANGUAGE ISSUE** refers to the inadequacy in the variety of human natural languages supported by the LLM open-source software. If the software could support a broader range of human languages, it would significantly enhance user experience. For example, a user asked the gpt4all team to “*support Chinese model*” (gpt4all #1199).

Interpretation: *Answer Issue* is a kind of quality issue related to answers generated by LLM open-source software. It is evident that the most prominent concerns in this category are **POOR-QUALITY ANSWERS** (35) and **INCORRECT ANSWERS** (28). *Answer Issue* arises due to the limitations of LLMs, which are unable to provide satisfactory answers. *Answer Issue* is also related to the way users formulate their questions. Some users do not follow the template specified by the development team when posing their questions, leading to responses that do not meet their expectations.

3.1.5. Performance Issue (7.14%)

Performance Issue refers to issues concerning the performance of LLM open-source software. This category encompasses the following four types.

- **SLOW RESPONSE** refers to the issues in which LLM open-source software shows a delayed response to requests. The response time significantly impacts the user experience. Sometimes, practitioners even encounter timeouts when waiting for answers. For example, a user reported that the application, which enables users to run and manage LLMs on their local machines, “*got stuck*” when running it and had to “*quit manually*” (ollama #1382).
- **PERFORMANCE OPTIMIZATION ISSUE** refers to the discussions on how to enhance the performance of LLM open-source software. Performance optimization of LLM open-source software has been a concern of practitioners. For example, one user reported that “*our deployment gives only 50 English words in 6 seconds*”, and the user inquired with the development team about how to optimize the output token per second (FastChat #1041).
- **UNSTABLE PERFORMANCE** refers to the instability of the performance of LLM open-source software under varying conditions. The performance of LLM open-source software may deteriorate due to longer execution time, or significant performance discrepancies may arise when execute on different devices or software environments. For example, a user reported that the software “*started lagging when it got past 3 lines and can take up to a minute to complete*”, which indicates that the response time of the software decreased over time (text-generation-webui #1542).

- **PERFORMANCE EVALUATION ISSUE** refers to the discussions on how to evaluate the performance of LLM open-source software. Practitioners attempt to discuss and establish a unified set of evaluation criteria to measure the performance of LLM open-source software. For example, a user wanted to know the performance of the model “*compared to vicuna-13b-v1.3 model*” and the ranking of this model in the Hugging Face (HF) community, which reflects concerns of users about the performance evaluation of the model (FastChat #2152).

Interpretation: *Performance Issue* mainly stems from the inadequacies and instability of the model performance. Due to SLOW RESPONSE (27) and UNSTABLE PERFORMANCE (14) of LLM open-source software, PERFORMANCE OPTIMIZATION ISSUE (26) and PERFORMANCE EVALUATION ISSUE (4) have been great concerns of practitioners. This category of issues reflect dissatisfaction of users with the response speed and performance stability of LLM open-source software.

3.1.6. Code Issue (6.74%)

Code Issue refers to the programming issues of LLM open-source software. This category encompasses the following five types.

- **FUNCTION ISSUE** refers to issues concerning the implementation of functions of LLM open-source software, such as the definition of the parameter list of the function, the naming of the function, issues within the code logic of the function, code syntax errors. For example, a user reported that the method “*max_marginal_relevance_search()*” was not implemented”, which led to the failure of word embedding (langchain #10059).
- **CLASS ISSUE** refers to issues concerning the implementation of classes of LLM open-source software, which covers issues about methods and parameters of a class, as well as defining new classes for specific requirements. For example, a user complained that “*it is impossible for anything outside of Microsoft.SemanticKernel.Abstractions to actually implement this abstraction*”, and asked the development team to modify the class “*IKernel*”, in order to improve the data extraction abilities of models (semantic-kernel #3195).
- **OBJECT ISSUE** refers to the issues with the instantiation and management of objects within LLM open-source software, which includes issues related to the manipulation of existing objects, management of object life cycles. For example, a user reported that “*property ‘id_hash_keys’ of the ‘Document’ objects cannot be set*”, which led to the failure of metadata comparison (haystack #1920).
- **CODE LANGUAGE SUPPORTING ISSUE** refers to requirements to support more programming languages

in LLM open-source software. For example, a user asked the langchain team “*to support the kotlin language*” in order to ensure “*the safety and reliability of the software*”, which is an open-source framework for building LLM applications (langchain #4963).

- **CODE STYLE STANDARDIZATION** refers to the discussion of normalizing coding styles. For example, a user suggested to “*standardize our code*” of the software, which is used for building end-to-end question-answering systems and retrieval-augmented generation models (haystack #2022).

Interpretation: *Code Issue* highlights the programming problems within LLM open-source projects. These concerns range from the definition and implementation of functions to issues in object-oriented programming and problems with code style standardization. FUNCTION ISSUE (47) is the most common type of *Code Issue*, as LLM open-source software involves complex algorithms, making the implementation and maintenance of functions challenging for developers.

3.1.7. Installation Issue (4.93%)

Installation Issue refers to the issues that arise during the installation of LLM open-source software. This category encompasses the following two types.

- **SETUP ERROR** refers to the issues encountered by practitioners during the installation or initialization process of LLM open-source software. For example, a user reported that the software, which is a localized open-source LLM, was “*not working on Windows after fresh installation*” (gpt4all #400).
- **DOWNLOAD ERROR** refers to the issues that arises during the process of downloading LLM open-source software. For example, a user reported that “*the download speed falls to 100 KB or something*” after 5% when downloading the localized LLM (gpt4all #734).

Interpretation: *Installation Issue* reflects the errors and anomalies that occur during the download and initialization of LLM open-source software. LLM open-source projects require compatibility with specific hardware, operating systems, and machine learning libraries. SETUP ERROR (38) as the dominant type of *Installation Issue* highlights the demands that LLM open-source software places on the deployment environment.

3.1.8. Documentation Issue (4.73%)

Documentation Issue refers to issues with the writing and management of documentation related to LLM open-source projects. This category encompasses the following five types.

- **MISSING DOCUMENTATION** reflects a lack of relevant documentation in LLM open-source projects, which leads to barriers for practitioners in developing and using the software. For example, a user suggested the FastChat team “*adding tutorials to increase token*

limits for self-deployment” in order to let the model be able to generate longer answers (FastChat #638).

- **DOCUMENTATION CONTENT ERROR** reflects that some parts of the documentation of LLM open-source projects contain inaccuracies or errors. Documentation content does not align with reality or there are inconsistencies among different parts of the documentation. For example, a user posted the screenshots of two documents that introduce the evaluation results of the models and asked “*which one is correct*”, showing inconsistency between the two documents (FastChat #1782).
- **OUT OF DATE DOCUMENTATION** reflects that certain portions of the documentation of LLM open-source projects are outdated. It shows that during the processes of software development, some documents have not been updated, resulting in outdated documentation which may lead to confusion among practitioners. For example, a user reported that “*documentation linked in official article is outdated*”, which contains information on model testing (semantic-kernel #4038).
- **INCOMPLETE DOCUMENTATION** reflects that certain parts of the documentation of LLM open-source projects are not complete. For example, a user asked, “*can you also suggest how/what to modify this model to run on a single/double GPU as I have access to those*”, which indicates that key information is missing from the documentation of the LLM open-source project (dolly #8).
- **UNCLEAR DOCUMENTATION** refers to the situation that certain documentation of LLM open-source projects provides ambiguous or obscure descriptions. Unclear descriptions lead to comprehension challenges, creating difficulties for practitioners. For example, a user was quite confused about the documentation related to the model parameters and asked “*could you please make this more descriptive*” (autogen #728).

Interpretation: *Documentation Issue* covers the issues in the documentation of LLM open-source projects, which lead to challenges in understanding, developing, and using of LLM open-source software for practitioners. MISSING DOCUMENTATION (13), DOCUMENTATION CONTENT ERROR (12), and OUT OF DATE DOCUMENTATION (11) are the top three types of *Documentation Issue*. This category of issues reflects that developers do not regularly review and update the documentation for LLM open-source projects.

3.1.9. Configuration Issue (4.53%)

Configuration Issue refers to issues concerning the configuration of LLM open-source software. This category encompasses the following two types.

- **CONFIGURATION SETTING ERROR** refers to the issues arising from setup settings of LLM open-source projects, including environmental variables, system configurations, component configurations, and machine learning library configurations. For example, a user received a message that “*Ray Dependency Conflict*” while configuring a distributed computing framework named Ray (haystack #1657).
- **CONFIGURATION FILE ERROR** refers to issues concerning configuration files of LLM open-source projects, including missing configuration files, reading and loading errors of configuration files. For example, a user was “*unable to gather the deployment name from the .env file*”, which led to the failure of deploying the localized model (semantic-kernel #1474).

Interpretation: *Configuration Issue* covers issues related to configuration encountered by practitioners during the deployment of LLM open-source software. LLM open-source software involves complex configuration options, leading to CONFIGURATION SETTING ERROR (30) during the configuration process. Moreover, if the project configuration tutorials are not sufficiently detailed or the guidance is unclear, users are more likely to make mistakes or fail to let LLMs meet their requirements when configuring LLM settings.

3.1.10. Network Issue (4.43%)

Network Issue refers to network environment issues concerning LLM open-source software, including issues with network connections and servers. This category encompasses the following two types.

- **NETWORK CONNECTION ISSUE** refers to the errors concerning network connections when using LLM open-source software, including network connection interruptions, connection timeouts, connection errors, and configuration issues related to network connection. For example, a user “*cannot connect to remote host*” of an open-source search engine, when performing word embeddings (langchain #7995).
- **SERVER ISSUE** refers to the issues in the servers of LLM open-source software, including server connection anomalies, server out of service, and server optimization problems. For example, a user reported a “*langchain-server error*” when preprocessing dataset of the model (langchain #822).

Interpretation: *Network Issue* covers the network-related issues and failures encountered by practitioners during the installation and use of LLM open-source software. *Network Issue* can affect the operation of models and responses to requests. NETWORK CONNECTION ISSUE (32) reflects how unstable network environments and network restrictions prevent LLM open-source software from properly responding to user requests. SERVER ISSUE (12) highlights the insufficient performance of servers hosting LLM open-source software,

their inadequate load-handling capacity when faced with heavy requests, and poor maintenance of server configurations (e.g., firewall rules).

3.1.11. Memory Issue (3.92%)

Memory Issue refers to issues concerning memory management and the data storage in memory when using LLM open-source software. This category encompasses the following two types.

- **OUT OF MEMORY** refers to the situation where available memory is exhausted and no additional memory can be allocated for LLM open-source software, which indicates that the computational and storage resources required to run LLM open-source software are substantial. For example, a user received an error message that “*OutOfMemoryError: CUDA out of memory. – train dolly v2*” (dolly #100).
- **STORAGE ISSUE** reflects the data storage issues in memory encountered by practitioners of LLM open-source software, which primarily involve issues related to memory allocation strategies, anomalies or failures in data storage. For example, a user reported that “*memory allocation error occurred*” when trying to “*inference on the given model*” (unilm #729).

Interpretation: *Memory Issue* covers anomalies in the allocation and management of system memory during the operation of LLM open-source software. An LLM contains billions of parameters, and deploying and running LLM open-source software requires plenty of memory. *Memory Issue* reflects the substantial demand for memory resources by LLM open-source software, which poses a challenge for practitioners with limited resources.

3.1.12. Prompt Issue (3.62%)

Prompt Issue refers to issues concerning the prompts input into LLM open-source software and the process of inputting prompts. This category encompasses the following two types.

- **PROMPT FAILURE** refers to issues encountered when inputting prompts into LLM open-source software, including exceeding the length limit of prompts, using an invalid prompt data type, containing invalid characters in prompts, or errors when uploading prompts. For example, a user reported that “*the error of ‘valid_languages’ occurred*” when uploading a PDF document as a prompt (haystack #874).
- **PROMPT PARSING ISSUE** refers to abnormalities and errors encountered by practitioners of LLM open-source software when parsing prompt content. For example, a user reported that the software “*only recognized the first four rows of a CSV file*” when chatting with the model (langchain #3621).

Interpretation: *Prompt Issue* covers issues that arise when inputting prompts into LLM open-source software. **PROMPT**

FAILURE is the largest type (28), which reflects the restrictions of LLM open-source software regarding the file format and content of the prompts.

3.1.13. Security Issue (3.52%)

Security Issue refers to security problems or vulnerabilities in LLM open-source software, including flaws, weaknesses, or errors within the system that could be exploited by malicious users to compromise its security. This category encompasses the following five types.

- **USER PERMISSION ISSUE** refers to issues regarding user permissions of LLM open-source software. Many users have expressed concerns regarding user permissions, questioning whether user permissions are either insufficient or excessive. For example, a user had “*no right to download checkpoint*”, a file that stores the weights and state of the model (FastChat #94).
- **ACCESS CONTROL ISSUE** refers to issues concerning the restriction of user access to data and system resources in LLM open-source software. The discussions on **ACCESS CONTROL ISSUE** aim to ensure the security and integrity of the software. For example, a user reported that “*it seems that GPT4ALL disallows*” the interaction between the local webpage and the model, “*and the access is blocked by CORS*” (gpt4all #941).
- **ILLEGAL INSTRUCTION** refers to the utilization of unauthorized instructions during the operation of LLM open-source software, which may potentially lead to security issues. For example, a user received a message that “*thread 5 ‘llm thread’ received signal SIGILL, Illegal instruction*”, which shows that an illegal instruction made the core dump (gpt4all #378).
- **PRIVACY ISSUE** refers to the risk of privacy breaches encountered by users when using LLM open-source software. For example, a user complained that “*different users share the same chat history*” when chatting with the model, which shows the concern of users about their privacy (text-generation-webui #2950).
- **MALWARE ISSUE** refers to the potential presence of malicious software attacks in LLM open-source software, which adversely affects the security of LLM open-source software. For example, a user reported that “*Windows defender claims to have found a backdoor in pytorch_model.bin*” (text-generation-webui #456).

Interpretation: *Security Issue* covers various security-related problems in LLM open-source software. As LLM open-source software becomes widespread, any developer can access and modify the source code or redeploy the LLM open-source software, which presents security challenges to practitioners, such as **USER PERMISSION ISSUE** (16), **PRIVACY ISSUE** (4), and **MALWARE ISSUE** (2). The proportion of *Security Issues* in LLM open-source software is currently

relatively small, but they cannot be overlooked, as these issues may lead to malicious exploitation of the software system, affecting the overall security of the software.

3.1.14. Graphical User Interface (GUI) Issue (2.72%)

GUI Issue refers to issues concerning the user interface of LLM open-source software. GUI serves as the channel through which users interact with the software and significantly impacts the user experience. This category encompasses the following two types.

- **GUI DESIGN ISSUE** refers to issues concerning the GUI design of LLM open-source software. A well-designed GUI enhances user experience, making optimization of GUI design an aspect worthy of attention. For example, a user suggested the development team “*activating extensions to appearing in GUI*” that allow users selecting chat models more easily on their own (text-generation-webui #2638).
- **GUI FAILURE** refers to the malfunction of the GUI in LLM open-source software, which emphasizes the maintenance required for the GUI. For example, a user reported that “*‘File Upload’ button not working*” when chatting with the model (ChatDev #227).

Interpretation: The design and maintenance of the GUI in LLM open-source software are closely tied to user experience, and consequently *GUI Issue* is a concern for practitioners. GUI DESIGN ISSUE (21) is the most common *GUI Issue*. This type of issues reflects that the GUI design of LLM open-source software involves complex interactions that are not user-friendly and negatively impact the user experience. Developers may focus more on implementing functionality that uses LLMs and overlook the detailed design of the user interface. As a result, insufficient attention has been given to the testing and maintenance of the GUI, leading to *GUI Issues*.

3.1.15. Database Issue (2.41%)

Database Issue refers to issues concerning the database of LLM open-source software. This category encompasses the following three types.

- **DATABASE QUERY ISSUE** refers to the issues concerning database queries in LLM open-source software. For example, a user reported that “*the table never changes, even if I change the custom_query like changing the fields*” when the user wanted to retrieve metadata for data preprocessing (haystack #685).
- **DATABASE CONNECTION ISSUE** refers to the issues regarding the connection between the database and LLM open-source software, such as database connection interruptions. For example, a user reported that the connection to the database was lost when “*resetting the prompt template*” (langchain #2396).
- **DATABASE DESIGN ISSUE** refers to the issues concerning the design of database tables in LLM open-source software. For example, a user said that the

number of fields in the “*onboarding*” database table is insufficient to describe the status of the agents, and the table should at least include the following fields: “*user_id, onboarding_A, onboarding_B_1, onboarding_B_2, onboarding_B_3*” (quivr #1328).

Interpretation: *Database Issue* covers various problems related to the databases in LLM open-source software, which can affect the data storage of LLM open-source software. DATABASE QUERY ISSUE (16) is the dominant type of *Database Issue*, which reflects the difficulties posed by the large-scale data within LLM open-source software, leading to inefficiencies in query execution and inaccuracies in query results.

3.2. Category of Causes (RQ2)

3.2.1. Results

As mentioned in Section 2.4.2, not all closed issues have an associated cause that can be identified. As a result, we identified a total of 559 causes, which were extracted from 56.2% of the 994 issues, and categorized into 11 categories as presented in Table 5. The results show that the most prominent cause is *Model Problem* (18.6%), followed by *Configuration and Connection Problem* (17.7%), and *Feature and Method Problem* (16.6%). The examples of each category of cause with their counts and proportions are also provided in Table 5. It should be noted that certain categories of issues can potentially be the causes of other issues. For example, MODEL COLLABORATION ISSUE is a type of *Model Issue*, and it can also be the cause of other types of issue such as POOR-QUALITY ANSWER and BLANK OUTPUT.

- **Model Problem (MP)** refers to problems concerning the models integrated into LLM open-source software. As the core component of LLM open-source software, the models are the most frequent source of issues. It encompasses five types: (1) MODEL INCOMPATIBILITY: Incompatibility between models, between models and other components, and between models and hardware. For example, a user received a message that “*Torch is not compatible with CUDA*”, leading to the failure of the installation of the LLM open-source software (text-generation-webui #355). (2) MODEL TRAINING PROBLEM: Problems occurring when training models and problems with the training algorithms. (3) OUTDATED MODEL: Using obsolete versions of models, and (4) MODEL ARCHITECTURE PROBLEM: Optimization and refinement needed in model architecture implementations.
- **Configuration and Connection Problem (CCP)** refers to problems concerning incorrect settings or failures in establishing connections within LLM open-source software. It encompasses three types: (1) INCORRECT CONFIGURATION: Failure of configuring LLM open-source software correctly. For example, a user failed to operate the localized LLMs, and found that “*there are*

Table 5

Causes of issues about LLM open-source projects

Category	Example (Extracted Cause)	Count	%
Model Problem (MP)	<i>My first guess would be that you are running an ancient version of Transformers. (text-generation-webui #1650)</i>	104	18.6%
Configuration and Connection Problem (CCP)	<i>This looks like a fastchat issue. From the error above, there might be python environment issues. (autogen #207)</i>	99	17.7%
Feature and Method Problem (FMP)	<i>Providing validation can be extremely useful to users, especially less experienced ones, or users with very complex pipeline designs. (haystack #1981)</i>	93	16.6%
Parameter Problem (PP)	<i>The error message indicates that the 'api_url' argument is no longer accepted by the 'anthropic.Client' initializer. (langchain #6883)</i>	72	12.9%
Low Performance Problem (LPP)	<i>Although the results converged after only a few epochs, it still is very slow. (unilm #144)</i>	42	7.5%
Component Problem (CP)	<i>Is there a dedicated document loader tool in AutoGen, ... (autogen #607)</i>	36	6.4%
Incorrect Operation (IO)	<i>If you try to import the PDFToTextConverter without having the [ocr] option installed, it will give you an error. (haystack #4189)</i>	26	4.7%
Hardware and Software Environment Incompatibility (HSEI)	<i>Looks like I'm missing the AVX instruction on my CPU. (gpt4all #378)</i>	22	3.9%
Resource Problem (RP)	<i>In this case it's OS memory; I think 64GB is actually not enough. (dolly #194)</i>	20	3.6%
Documentation Problem (DP)	<i>You must follow the three steps: ... (FastChat #186)</i>	20	3.6%
Security Problem (SP)	<i>Langchain agents can be hijacked while searching internet via injection prompts. (langchain #4102)</i>	15	2.7%

python environment issues” after testing “FastAPI”, a framework for building Web APIs (autogen #207). (2) NETWORK CONNECTION PROBLEM: Issues arising from disruptions or failures in network connections of LLM open-source software, and (3) DATABASE CONNECTION PROBLEM: Problems in establishing connections with the database of LLM open-source software.

- **Feature and Method Problem (FMP)** refers to problems concerning the functionality of LLM open-source software and its implementation methods. It encompasses two types: (1) MISSING FEATURE: The absence of one or more essential functions within LLM open-source software. For example, a user reported that the loss of a key feature of the retriever led to the issue that the model often “*gives non-sense answers*” (haystack #883), and (2) INCORRECT METHOD DEFINITION AND IMPLEMENTATION: Flaws in the logic, definition, and implementation of functions and methods of LLM open-source software.
- **Parameter Problem (PP)** refers to problems concerning parameters within LLM open-source software. It encompasses two types: (1) PARAMETER SETTING AND PASSING PROBLEM: Problems on the definition, assignment, and subsequent passing of parameters in LLM open-source software. For example, a user explained that “*the embedding_function in model.encode() is returning*” the array type not supported by the search engine, leading to the failure of the word embedding (langchain #2016), and (2) MISSING PARAMETER: The absence of critical parameters in model training scripts, model hyperparameters, model configuration parameters, and inference scripts.
- **Low Performance Problem (LPP)** refers to deficiencies in the performance of LLM open-source software. It encompasses five types: (1) LOW THROUGHPUT: Low data processing capability in LLM open-source

software. For example, a user explained that “*Whisper API limits upload to 25MB*”, leading to the failure of preprocessing audio files (quivr #3). (2) SLOW RESPONSE: Response latency or timeout by LLM open-source software. (3) LOW SCALABILITY: Poor scalability of LLM open-source software, limiting its ability to handle increasing loads. (4) LOW STABILITY: Runtime sensitivity of LLM open-source software to the software or hardware environment, and (5) LOW RESOURCE UTILIZATION: Inefficient resource utilization of LLM open-source software.

- **Component Problem (CP)** refers to the problems among the various components of LLM open-source software. It can be categorized into three types: (1) COMPONENT INCOMPATIBILITY: Components are incompatible with other components within LLM open-source software. For example, the LangChain team explained that “*ZeroShotAgent does not support multi-input tool Calculator*”, which led to the failure of operating the model (langchain #3781). (2) MISSING COMPONENT: The absence of crucial components within LLM open-source software, and (3) OUTDATED COMPONENT: Outdated versions of components within LLM open-source software.
- **Incorrect Operation (IO)** refers to the improper actions taken by practitioners of LLM open-source software. It can be categorized into three types: (1) INCORRECT OPERATION PROCEDURE: Wrong steps in operating LLM open-source software. For example, the development team found that the user who raised the issue “*had skipped the llama to hugging face weight conversion step*”, leading to the failure of deploying the LLM open-source software (FastChat #236). (2) INCORRECT METHOD: Utilizing incorrect functions for feature implementation of LLM open-source software, and (3) INCORRECT INSTRUCTION: Errors in instructions used to operate LLM open-source software.

- **Hardware and Software Environment Incompatibility (HSEI)** refers to the problems encountered on the devices that LLM open-source software is deployed. It can be categorized into two types: (1) **HARDWARE INCOMPATIBILITY**: Incompatibility between hardware and LLM open-source software. For example, a user explained that “*I do not think 1080ti support 4int instruction set*”, leading to the failure of deployment of the LLM open-source software (gpt4all #378), and (2) **SOFTWARE ENVIRONMENT INCOMPATIBILITY**: Incompatibility between the software environment (e.g., operating system, virtual machine) and the LLM open-source software.
- **Resource Problem (RP)** refers to the problems concerning the resources used by LLM open-source software. It can be categorized into two types: (1) **RESOURCE SHORTAGE PROBLEM**: Insufficient system resources of LLM open-source software. For example, a user reported that “*64GB is actually not enough*” for training the model, and memory shortage led to the failure of model training (dolly #194), and (2) **RESOURCE ALLOCATION PROBLEM**: Unreasonable system resource allocation of LLM open-source software.
- **Documentation Problem (DP)** refers to the problems within the documentation of LLM open-source projects. It can be categorized into two types: (1) **POOR DOCUMENTATION**: The documentation of LLM open-source projects is incomplete, inaccurately described, lacking in detail, overly vague, or even nonexistent. For example, a user complained that “*add tutorials to increase token limits for Self-Depoly*”, as the loss of key documentation posed challenges for users in deploying the model (FastChat #638), and (2) **OUTDATED DOCUMENTATION**: Documentation of LLM open-source projects becomes obsolete due to lack of updates or revisions.
- **Security Problem (SP)** refers to problems concerning the security of LLM open-source software. It can be categorized into four types: (1) **ACCESS PROBLEM**: Failure in security authentication when using LLM open-source software. For example, a user did not set an “*Elasticsearch client instance, index name, and embedding object*” to obtain access permissions, resulting in the failure of environment configuration for the model (langchain #1865). (2) **ILLEGAL INSTRUCTION**: Using unauthorized instructions to operate LLM open-source software. (3) **USER PERMISSION PROBLEM**: Insufficient or excessive permission for accessing and operating resources and functions in LLM open-source software, and (4) **MALWARE PROBLEM**: Attacks on LLM open-source software by malware.

3.2.2. Causes to Issues Mapping

Table 6 illustrates the mapping between the *Issue* categories and *Cause* categories in developing and using LLM open-source software, along with their distribution, using abbreviations to represent each category of *Cause*. For example, “MP” represents *Model Problem*. The full names of all cause categories are provided in the note of Table 6.

Over three quarters (78.3%) of *Parameter Issues* are identified with associated causes, reaching the highest rate among all categories of issues, and *Parameter Issues* are mainly caused by MP (24).

Only 12.8% of *Documentation Issue* are identified with corresponding causes, having the lowest rate among all categories of issues. There are only a few cases of *Documentation Issues* have their causes, and FMP (3) is identified as the most frequent cause.

The largest category among all the issues is *Model Issue*. 50.4% of *Model Issues* are identified with the corresponding causes. *Model Issue* mainly originates from CCP (25), FMP (21), and PP (22).

For *Component Issue*, causes are identified in 52.2% of cases. *Component Issue* is mainly caused by MP (15).

For *Answer Issue*, causes are identified in 53.7% of cases. MP (9) and CCP (9) are main causes leading to *Answer Issue*.

For *Performance Issue*, causes are identified in 59.2% of cases. FMP (15) is identified as the most frequent cause of *Performance Issue*.

For *Code Issue*, causes are identified in 65.7% of cases. *Code Issue* is mainly induced by FMP (11).

For *Installation Issue*, causes are identified in 61.2% of cases. *Installation Issue* mainly originates from CCP (14).

For *Configuration Issue*, causes are identified in 60% of cases. *Configuration Issue* is mainly caused by CCP (8).

For *Network Issue*, causes are identified in 59.1% of cases. *Network Issue* is mainly brought about by CCP (7).

For *Memory Issue*, causes are identified in 66.7% of cases. *Memory Issue* is mainly induced by MP (9).

For *Prompt Issue*, causes are identified in 63.9% of cases. LPP (5) is identified as the most frequent cause of *Prompt Issue*.

For *Security Issue*, causes are identified in 51.4% of cases. *Security Issue* is mainly caused by FMP (3), PP (3), IO (3), and HSEI (3).

For *GUI Issue*, causes are identified in 25.9% of cases. *GUI Issue* mainly originates from CCP (3).

For *Database Issue*, causes are identified in 54.2% of cases. *Database Issue* is mainly brought about by MP (4).

3.2.3. Interpretation

Frequency of the causes: MP is the most frequent factor contributing to issues. As core functional components of LLM open-source software, the fact that MP is the most frequent cause of issues shows that the challenges faced by the models remain formidable. *Model Issue* stems from various factors such as model architecture, updates, training, and resource requirements, making the use and maintenance of LLMs challenging for practitioners. In addition, CCP

Table 6

Mapping between issue categories (vertical) and cause categories (horizontal)

	MP	CCP	FMP	PP	LPP	CP	IO	HSEI	RP	DP	SP
Model Issue	0	25	21	22	15	10	2	8	10	6	4
Component Issue	15	5	2	11	3	0	4	2	0	3	3
Parameter Issue	24	13	16	0	5	6	3	1	0	3	1
Answer Issue	9	9	4	8	5	1	5	1	0	1	1
Performance Issue	14	2	15	3	0	2	1	1	2	0	2
Code Issue	8	6	11	8	2	3	1	0	3	1	1
Installation Issue	6	14	0	0	0	5	2	1	0	1	1
Documentation Issue	0	0	3	1	1	1	0	0	0	0	0
Configuration Issue	5	8	3	4	0	2	1	0	1	3	0
Network Issue	4	7	4	5	1	1	0	2	0	0	2
Memory Issue	9	3	4	3	1	1	0	1	4	0	0
Prompt Issue	4	3	2	2	5	2	2	1	0	2	0
Security Issue	2	1	3	3	1	2	3	3	0	0	0
GUI Issue	0	3	2	0	0	0	1	1	0	0	0
Database Issue	4	0	3	2	3	0	1	0	0	0	0

Full names of every cause category abbreviation: MP: Model Problem; CCP: Configuration and Connection Problem; PP: Parameter Problem; FMP: Feature and Method Problem; LPP: Low Performance Problem; CP: Component Problem; IO: Incorrect Operation; HSEI: Hardware and Software Environment Incompatibility; RP: Resource Problem; DP: Documentation Problem; SP: Security Problem.

and FMP also significantly contribute to the issues. These types of causes are general problems in the development of LLM open-source software. The process of integrating LLMs into software to implement functionality is highly complex. When selecting components, users may encounter incompatibility between components, between components and models, and between components and the software or hardware environment. In the implementation of functionalities, design flaws of functions and classes can arise. In addition, LLM open-source software contains complex algorithms, which presents challenges in implementing the features of LLM open-source software. Although each of the remaining eight categories of causes of the issues accounts for a relatively minor proportion, they also cannot be ignored. For example, during the definition and settings of parameters, there may be problems related to incorrect parameter settings, such as improper values, improper data type, missing parameters, and redundant parameters. PP reflects that the large-scale parameters of LLMs introduce difficulties in the development and maintenance of LLM open-source software.

Mapping of causes to issues: Nearly a quarter (24.0%) of *Model Issues* are caused by CCP. Model version updates and differences in model types may lead to incompatibility between the components and models. When using different models or frameworks, or after model updates, parameter formats or inference mechanisms may be changed, which can also results in incompatibility with other components. 20.2% of *Model Issues* are induced by FMP, which reflects the complexity of the design and implementation of features of LLM open-source software. The accuracy of responses, the performance of models, and execution of training or inference in LLMs rely on the proper implementation of functionalities. If the functional design is flawed or the implementation method is incorrect, the performance of models will be adversely affected. 21.2% of *Model Issues* originate from PP, and 33.3% of *Parameter Issues* are caused by MP. The design and execution of the model rely on parameter settings, if the parameters are not set properly, the

model may fail to function correctly. LLMs, with their large-scale parameters, make parameter settings more difficult, and the design flaws and limitations of models (such as restrictions on parameter length) bring specific constraints to parameter settings, leading to *Parameter Issues*.

3.3. Category of Solutions (RQ3)

3.3.1. Results

As mentioned in Section 2.4.2, not all closed issues have an associated solution that can be identified. As a result, we identified a total of 798 solutions, which were used to address 80.3% of the 994 issues, and categorized into seven categories as presented in Table 7. The results show that *Optimize Model* (OM) is the most commonly utilized solution, accounting for 29.6% of the total solutions; closely followed by *Adjust Parameter* (AP), *Adjust Configuration and Operation* (ACO), and *Optimize Component* (OC), accounting for 19.4%, 18.0%, and 17.2% respectively. The examples of each category of solution with their counts and proportions are provided in Table 7.

- *Optimize Model* (OM) refers to the process of solving problems by fine-tuning or refining the LLMs. It is also the most widely accepted solution for addressing problems. It can be categorized into five types: (1) OPTIMIZE MODEL ARCHITECTURE: Improving the architectural design and implementation of LLMs. For example, a user suggested that “it is better to convert that model to llama CPP ggml and use it with the wrapper” to address the issue of the inability of the model to operate on Intel Core i7 CPU. (text-generation-webui #1233). (2) OPTIMIZE MODEL TRAINING: Enhancing the training methods of LLMs, such as improving training algorithms and adjusting relevant process controls when training models. (3) UPDATE MODEL: Updating the models by replacing old versions with newer ones. (4) USE STABLE VERSION OF MODEL: Utilizing a version of the LLM that offers stable functionality and performance to ensure stability, and (5) OPTIMIZE DATASET: Selecting

datasets, which are more widely used by practitioners, for training, fine-tuning, and evaluating the models integrated into LLM open-source software.

- *Adjust Parameter (AP)* refers to resolving issues by adjusting parameters. Parameters play a crucial role in software systems, especially in LLM open-source software, which involves billions of parameters. Therefore, it accounts for a significant proportion. It can be categorized into five types: (1) **MODIFY PARAMETER**: Modifying the definition and values of parameters within LLM open-source software to address the problem. For example, a user suggested “*setting ‘infer_tokenizer_classes = True’ when initializing DPR (Dense Passage Retrieval)*” to address the issue of freezing during the preprocessing of the training dataset (haystack #822). (2) **REMOVE PARAMETER**: Removing redundant parameters within LLM open-source software. (3) **ADD PARAMETER**: Adding parameters to fill in missing parameters or adding parameters to control new functionality for LLM open-source software. (4) **CONVERT PARAMETER TYPE**: Converting parameter types during parameter passing to match subsequent parameter processing of LLM open-source software, and (5) **ADD PARAMETER VALIDATION**: Adding parameter validation to ensure the correctness of functionalities within LLM open-source software.
- *Adjust Configuration and Operation (ACO)* refers to resolving issues by modifying software configurations and user operations. Proper configuration operation are crucial for the normal functioning of LLM open-source software. It can be categorized into four types: (1) **MODIFY INCORRECT OPERATION PROCEDURE**: Adjusting operation procedures by following related documentation and tutorials to operate LLM open-source software. For example, the development team suggested users “*initializing an empty tool list first, then adding a custom tool*” to address the custom tool failure (langchain #2569). (2) **MODIFY CONFIGURATION**: Adjusting configurations of LLM open-source software, including environmental settings, initialization configurations. (3) **RELOAD PROJECT**: Reloading or restarting LLM open-source software, and (4) **MODIFY INCORRECT INSTRUCTION**: Correct errors in instructions for operating LLM open-source software.
- *Optimize Component (OC)* refers to the adjustment and enhancement of various components within LLM open-source software. It can be categorized into six types: (1) **ADD NEW COMPONENT**: Introducing new components to implement new functionalities for LLM open-source software. For example, a user announced that “*I created the ‘chatmemorysources’ container*” to address the failure of uploading the documents when chatting with the model (semantic-kernel #1402). (2) **ADJUST COMPONENT INTEGRATION STRATEGY**: Modifying the strategies used to integrate various components within LLM open-source software, such as component decoupling and adjusting interfaces between components, to enhance the stability, scalability, and performance of LLM open-source software. (3) **OPTIMIZE COMPONENT FUNCTIONALITY**: Optimizing specific component functionalities within LLM open-source software to address corresponding issues. (4) **USE STABLE VERSION OF COMPONENT**: Using stable versions of components to ensure relatively stable functionality and performance of LLM open-source software. (5) **UPDATE COMPONENT**: Incorporating newer versions of components to iterate functionalities of LLM open-source software, and (6) **REMOVE COMPONENT**: Eliminating redundant, outdated, inefficient, incompatible, and vulnerable components of LLM open-source software.
- *Optimize System Resource Management (OSRM)* refers to adjusting the strategies of managing system resources for operating LLM open-source software. It can be categorized into four types: (1) **OPTIMIZE NETWORK CONNECTION MANAGEMENT**: Adjusting network connection settings and protocol implementations to ensure that LLM open-source software runs in an optimal network environment. For example, a user suggested that “*you can stop the generation by posting to the server*” instead of using Server-Sent Events (SSE) to address the issue that “*closing http socket does not cancel generation*” (text-generation-webui #4521). (2) **OPTIMIZE MEMORY MANAGEMENT**: Adjusting memory allocation and management strategies for LLM open-source software. (3) **OPTIMIZE MESSAGE MANAGEMENT**: Adjusting the messaging management mechanism within LLM open-source software, including aspects such as pipe communication and message passing, and (4) **OPTIMIZE PROCESS MANAGEMENT**: Optimizing process management (e.g., concurrent processing) and resource allocation strategies within LLM open-source software to reduce waiting time, minimize resource waste, and prevent deadlocks.
- *Modify Documentation (MD)* refers to altering and refining the documentation of LLM open-source projects. It can be categorized into three types: (1) **MODIFY DESCRIPTION**: Modifying descriptions in the documentation of LLM open-source projects, involving correcting wrong, vague, outdated, and inconsistent descriptions within the documentation. For example, a user suggested “*separating all commands and text to improve readability*”, which can address the issue of unclear descriptions in the deployment tutorial of the LLM open-source project (haystack #1863). (2) **ADD DOCUMENTATION**: Adding new

Table 7

Solutions of issues about LLM open-source projects

Category	Example (Extracted Solution)	Count	%
Optimize Model (OM)	<i>I think its better to convert that model to llama CPP ggml and use it on CPU that way with the wrapper. (text-generation-webui #1233)</i>	236	29.6%
Adjust Parameter (AP)	<i>To fix this issue, you need to replace 'api_url' with 'base_url' in the 'Anthropic' class in your LangChain code. (langchain #6883)</i>	155	19.4%
Adjust Configuration and Operation (ACO)	<i>Have you tried cloning the model repository into your models directory and loading it that way? (text-generation-webui #3656)</i>	144	18.0%
Optimize Component (OC)	<i>We will also soon release a high throughput batching backend. (FastChat #1041)</i>	137	17.2%
Optimize System Resource Management (OSRM)	<i>The workaround currently is to manually increase the swap space on your drives to be at least 140GB, but you can set a minimum to like 16mb so its not always huge. (text-generation-webui #1647)</i>	63	7.9%
Modify Documentation (MD)	<i>In addition, let's make sure that the use case of DocumentRetrieval pipelines as mentioned in #1618 is properly documented. haystack #1863)</i>	54	6.8%
Change Hardware and Software Environment (CHSE)	<i>We will opt by changing the machine we're using to one with an NVIDIA Graphics Card. (haystack #1643)</i>	9	1.1%

documentation of LLM open-source projects to synchronize documentation with the software updates, and (3) RENAME DOCUMENTS: Modifying document names of LLM open-source projects that are ambiguous or contain illegal characters.

- *Change Hardware and Software Environment (CHSE)* refers to modifying and optimizing the hardware and software environment for running operating LLM open-source software. It can be categorized into two types: (1) *CHANGE HARDWARE*: Replacing hardware for configuring LLM open-source projects, such as CPU, GPU, or even the entire computer. For example, the haystack team announced that “we opt by changing the machine we are using to one with an NVIDIA Graphics Card” to address the failure of deploying the LLM open-source software (haystack #1643), and (2) *CHANGE SOFTWARE ENVIRONMENT*: Replacing software environment for operating LLM open-source projects, such as operating system, virtual machine, and container.

3.3.2. Solutions to Issues Mapping

Table 8 illustrates the mapping of LLM open-source projects related *Issue* categories to *Solution* categories, along with their distribution, using abbreviations to represent each category of solution. For example, “OM” represents *Optimize Model*. The full names of all solution categories are provided in the note of Table 8.

For *Code Issue*, 92.5% of cases are well addressed, reaching the highest resolution rate among all categories of issues. OM (25) is identified as the most frequent solution for resolving *Code Issues*.

For *Component Issue*, 70.7% of the cases are effectively addressed, reaching the lowest resolution rate among all categories of issues, and *Component Issues* are mostly solved by OC (27).

For *Model Issue*, which is the largest category among all categories of issues, 80.1% of cases have their effective solutions. *Model Issues* are mostly addressed by OM (88).

For *Parameter Issue*, 83.7% of the cases are well addressed. The most frequently used solution for *Parameter Issues* is AP (36).

For *Answer Issue*, 86.6% of cases are identified with their solutions. OM (20) is mostly employed solution to resolve *Answer Issues*.

For *Performance Issue*, 77.5% cases are well resolved. The most frequently employed solution for *Performance Issues* is OM (21).

For *Installation Issue*, 81.6% of cases have their effective solutions. ACO (17) is identified as the most frequently used solution to address *Installation Issues*.

For *Documentation Issue*, 74.5% of the cases are identified with corresponding solutions. The most frequently employed solution for *Documentation Issues* is MD (23).

For *Configuration Issue*, 75.6% of cases are effectively addressed. The most frequently employed solution to address *Configuration Issues* is ACO (10).

For *Network Issue*, 75% of cases are addressed by practitioners. The most frequently employed solution for *Network Issues* is AP (13).

For *Memory Issue*, 89.7% of cases are identified with their associated solutions. OM (11) is the most commonly applied solution to resolve *Memory Issues*.

For *Prompt Issue*, 77.8% of cases are resolved. The most frequently used solution to address *Prompt Issues* is OM (16).

For *Security Issue*, 77.1% of cases are resolved by practitioners. ACO is identified as the most commonly applied solution to (9) address *Security Issues*.

For *GUI Issue*, 77.8% of cases are well addressed by practitioners. The most commonly employed solution for *GUI Issues* is OC (14).

For *Database Issue*, 91.7% of cases are well resolved by practitioners. OC is identified as the most frequently employed solution for *Database Issues*.

3.3.3. Interpretation

Frequency of the solutions: OM is the most frequently employed solution for addressing issues. The models are the core functional components of LLM open-source software. Therefore, OM can directly enhance the performance of

Table 8

Mapping between issue categories (vertical) and solution categories (horizontal)

	OM	AP	ACO	OC	OSRM	MD	CHSE
Model Issue	88	20	37	24	13	7	4
Component Issue	8	14	12	27	1	3	0
Parameter Issue	15	36	13	4	5	4	0
Answer Issue	20	18	16	11	5	1	0
Performance Issue	21	11	3	12	7	1	0
Code Issue	25	14	8	6	6	3	0
Installation Issue	10	3	17	6	1	2	1
Documentation Issue	2	2	5	2	1	23	0
Configuration Issue	7	6	10	6	1	3	1
Network Issue	4	13	5	5	6	0	0
Memory Issue	11	8	2	3	8	1	2
Prompt Issue	16	3	0	5	3	1	0
Security Issue	6	2	9	2	3	4	1
GUI Issue	1	1	3	14	1	1	0
Database Issue	2	4	4	10	2	0	0

Full names of every solution category abbreviation: OM: Optimize Model; AP: Adjust Parameter; ACO: Adjust Configuration and Operation; OC: Optimize Component; OSRM: Optimize System Resource Management; MD: Modify Documentation; CHSE: Change Hardware and Software Environment.

models, including accuracy, response speed, and the capability to handle complex tasks. Furthermore, fine-tuning the models can ensure more stable performance across various hardware and software environments, thereby guaranteeing its effective operation under diverse conditions. In addition, AP, ACO, and OC are also widely accepted by practitioners as solutions for problem-solving. LLM open-source software contains numerous parameters, especially in the models. Therefore, adjusting parameter settings and corresponding values, and implementing appropriate parameter passing methods, are feasible and effective approaches. Proper configuration and operation are essential for any software to function correctly, and LLM open-source software is no exception. Consequently, ACO is an effective approach to problem-solving. Furthermore, LLM open-source software is composed of various components, so as a general solution, OC is also widely accepted by practitioners as a method for addressing the issues. Although each of the remaining three categories of solutions for addressing the issues accounts for a relatively minor proportion, they still cannot be ignored. For example, OSRM reflects that practitioners can address issues within LLM open-source software by focusing on system resources.

Mapping of solutions to issues: For *Code Issue*, effective solutions have already been implemented. Since issues in programming directly affect the proper functioning of LLM open-source software, especially issues related to the logic of functions. Therefore, *Code Issue* tends to receive significant attention and is usually addressed promptly. OM constitutes a significant portion of the solution categories in every *Issue* category and is the most prevalent solution category overall. It is due to the predominance of *Model Issue* and *Model Problem*, and LLMs are core function component of LLM open-source software. Regardless of the category of the issue, the performance of the model directly impacts the overall performance of LLM open-source software. Therefore, OM not only enhances the capability of models to address specific issues, but also improves the stability and efficiency of the entire software. Furthermore, *Model Issue* can be addressed through all categories of

solutions, which indicates that there are multiple approaches to resolving issues concerning LLMs. The resolution rate for all categories of issues has exceeded 70%, which indicates that issues within LLM open-source software have received widespread attention from practitioners and have been effectively addressed.

4. Implications

In this section, we discuss the implications for practitioners and researchers of LLM open-source projects based on the study results presented in Section 3.

4.1. Implications for Users of LLM Open-Source Software

Implication 1. Users of LLM open-source software can optimize input prompts to obtain more satisfactory answers from LLM open-source software.

Answer Issue ranks the fourth among all 15 categories of issues, and within *Answer Issue*, the dominant type is *Poor-quality Answer* (42.7%), which indicates a level of dissatisfaction among users with the responses provided by LLM open-source software. However, some techniques can be employed to improve the input prompts in order to obtain better responses. For example, users can utilize predefined templates to input prompts for obtaining more refined responses. A user suggested that “you can provide context like below as in the Alpaca data”, which indicates that users can mimic the format of training data by segmenting input prompts into multiple parts such as instructions, input, and response, and write the context according to the syntax and standards of the training data to achieve better responses (dolly #55). Furthermore, users can also divide input prompts that exceed the length limit into multiple segments and enter these segments separately, filter out sensitive vocabulary, or provide more specific scenarios to obtain better responses. For example, the langchain project provides several built-in text splitting tools, such as *CharacterTextSplitter*, which can divide a document into smaller chunks for input into LLMs,

and allows maintaining continuity between chunks by setting “*chunk_overlap*”.

Implication 2. Users can consider using LLM Development Frameworks as assistants for their software development, while remaining mindful of the potential problems and risks associated with LLMs.

Over half of the LLM open-source projects investigated in this study are frameworks for developing software by leveraging LLMs (see Table 2), which could lower the barrier for users to develop software systems with the assistance of LLMs. For example, the langchain project provides an open-source software framework for developing applications powered by LLMs, and it supports both open-source and commercial LLMs (such as GPT from OpenAI, Llama from Meta), allowing users of the langchain project to select the LLMs that fit their needs in software development. However, according to the results of our study, there are plenty of issues associated with generating code using LLMs. 46.3% of Model Issues, 71.7% of Component Issues, 65.2% of Parameter Issues, 53.7% of Answer Issues, and 63.4% of Performance Issues originate from LLM Development Frameworks, which indicates that using LLM open-source software for development might lead to many potential problems and risks. For example, Fu et al. (2025) examined the security risks of Copilot-generated code, and found that that 29.5% of Python and 24.2% of JavaScript code snippets contain security weaknesses. Users of LLM Development Frameworks need to be aware of such issues associated with LLMs and take appropriate measures to prevent insecure code merging into the code base.

4.2. Implications for Developers of LLM Open-Source Software

Implication 3. Developers of LLM open-source software should possess a basic set of knowledge of LLMs.

According to our study results, the following knowledge is required to develop and use LLM open-source software: (1) Developers need knowledge of configuring the development and runtime environments for LLM open-source software. According to our study results, *Configuration and Connection Problem* ranks the second among the causes of the issues, which indicates that practitioners need solid knowledge about environment configuration of LLM open-source software. For example, a contributor received a message that “*Package ‘tokenizers’ not found in index and Package ‘taggers’ not found in index*”, which indicates key packages are missing from the dependency environment when configuring the development environment for LLM open-source software (langchain #8419). (2) Developers need to understand basic text processing techniques to handle input and output texts. LLMs are a kind of PLMs, and their input and output are text. Therefore, when developing LLM open-source software, it is necessary to have a basic understanding of text processing techniques like tokenization, stemming,

and lemmatization, stop word removal. For example, a contributor reported that “*–no-stream is very slow, because it ignores the stop words filled into stopping_criteria*”, which indicates that operational efficiency of LLM open-source software can be enhanced after the removal of stop words (text-generation-webui #805). (3) Developers need to have a basic understanding of deep learning frameworks that LLMs (e.g., GPT4) are based on. Acquiring knowledge of deep learning frameworks can facilitate the fine-tuning and inference functionalities of LLMs. For example, a contributor received a message that “*RuntimeError: PyTorch is not linked with support for mps devices!*”, which indicates that practitioners need a basic understanding of PyTorch to deploy the software (FastChat #854).

Implication 4. Developers should use more efficient parameter fine-tuning methods and design more robust parameter validation mechanisms.

According to Table 6, it can be observed that 22 cases of *Model Issue* are caused by *Parameter Problem*, 24 cases of *Parameter Issue* are induced by *Model Problem*, which reflects that *Model Issues* and *Parameter Issues* in LLM open-source software are interrelated, highlighting the critical importance of parameter settings, fine-tuning, and validation for the stability and performance of models. Therefore, developers need to introduce more effective parameter tuning methods for LLMs to avoid potential negative impacts on the performance of models. Zhou et al. (2024) suggested employing efficient parameter-efficient fine-tuning (PEFT) methods such as LoRA, IA3, Adapter, and Prefix-Tuning for fine-tuning multimodal models, which may even outperform full fine-tuning (FFT). Additionally, developers (e.g., developers of LLMs, OpenAI) should design more robust parameter validation mechanisms to minimize *Model Issues* arising from incorrect parameter values and types. For example, Frantar and Alistarh (2023) proposed a method called SparseGPT, which significantly compresses LLMs by pruning a vast number of parameters. The method uses Hessian information to determine which weights should be pruned and which should be retained. During the pruning process, the pruning mask and weight updates are dynamically adjusted based on the error at each layer and Hessian information to validate whether the pruned parameters negatively affect the accuracy of the LLMs.

Implication 5. Developers need to be cautious when selecting models and components when developing LLM open-source software.

As mentioned in Section 3.1, LLM open-source software contains various components, and some LLM open-source software integrates multiple models, which are developed by different teams and communities. Combining these models and components into one software system may lead to incompatibilities, such as MODEL INCOMPATIBILITIES and COMPONENT INCOMPATIBILITIES. Additionally, updates of

these models and components can also lead to incompatibilities. Developers should carefully read the documentation of models and components to understand their supported features and provided interfaces. When introducing new versions of models and components in LLM open-source software, developers should carefully consider and explore the potential impact of these models and components on the compatibility within the existing system.

4.3. Implications for Researchers of LLM Open-Source Software

Implication 6. Researchers may consider to provide automated parameter adjustment tools and methods.

According to Table 6, it is obvious that *Model Issues* and *Parameter Issues* in LLM open-source software are interrelated. Proper parameter settings significantly affect the performance of LLMs. LLM open-source software is used by plenty of end-users, many of whom may not be proficient in parameter settings. However, LLMs contain billions of parameters (e.g., learning rate, batch size, and weight decay), manual adjustments is time-wasting and inefficient. Automatic tools can systematically search for the optimal parameter settings to ensure better model performance and to make it easier for practitioners to use LLM open-source software. For example, Falkner et al. (2018) proposed a hyperparameter tuning method called BOHB, which combines the advantages of Bayesian optimization and bandit-based methods to achieve both stable performance and rapid convergence to optimal settings.

Implication 7. Researchers may consider to provide methods to reproduce the problems occurred in LLM open-source software.

Due to the randomness of LLMs, among 226 unsolved issues, 38 issues show difficulties in reproducing the problems. Without accurately reproducing the problem within LLM open-source software, it can be difficult for practitioners to identify root causes of the issues, thereby hindering the resolution of the issues. For example, after several users reported that “*api_server runs too slowly*” when chatting with the model, some other users reported that “*I do not see that problem, really*”, which reflects difficulties in reproducing the problem (FastChat #1499), leading to the issue being unsolved.

Implication 8. Researchers may consider establishing standardized benchmarks to evaluate the impact of different configurations (e.g., model, dataset, hardware, and software environment) on the performance of LLMs.

According to Table 6, *Configuration and Connection Problem* is the most frequent cause to *Model Issues*, and according to Table 8, *Adjust Configuration and Operation* addresses 37 *Model Issues* among the 193 resolved *Model Issues*. Therefore, different configurations (e.g., model, dataset, hardware, and software environment) may impact

the performance of models when developing and using LLM open-source software. To better understand the influences of these configurations and improve model performance, researchers can establish standardized benchmarks for evaluating the impact of various configurations on the LLM performance. These benchmarks can facilitate comparing model performance under various configurations, providing a foundation for further optimization and enhancement of the performance of LLM open-source software.

Implication 9. Researchers may consider exploring how to better coordinate the collaboration between models and the interoperation between components to reduce incompatibilities within the system.

As mentioned in Section 3.2, *MODEL INCOMPATIBILITY* is the dominant type of *Model Problem*, and *COMPONENT INCOMPATIBILITY* is the largest type of *Component Problem*. In LLM open-source software, there are various components, and some LLM open-source software integrates multiple models. These models and components are mostly developed by different teams, which may lead to incompatibilities during the integration of these models and components, the collaboration of the models to answer user questions or complete tasks, as well as the interoperation of the components to run the LLM open-source software. By coordinating collaboration between models and interoperation between components, issues induced by incompatibilities can be reduced, thereby enhancing the extensibility of LLM open-source software, allowing developers to more easily introduce new models and add new components to implement new features and extend existing functionalities.

5. Threats to Validity

Construct validity: Due to the manual operation of data collection, labelling, sampling, and extraction in our study, there is a risk of personal bias on the data labelling and extraction results. To mitigate this risk and enhance the objectivity and reliability of our study results, we conducted pilot studies before each formal step to assess the effectiveness of methods and criteria among different researchers, which increased the construct validity of the study. Following each pilot and formal study, the first author engaged in discussions with the second and third authors to reach consensus on the data analysis results aligned with the RQs. Since the open coding procedure was carried out by the first author, we acknowledge the potential impact of overconfidence of the first author on the validity of the data extraction results. However, the first author had five years of experiences in software development two years of experience in using LLMs, which partially mitigates the potential threat to the validity of the data extraction results. Besides, we consider adopting automated or semi-automated tools as a future measure to mitigate potential biases in data selection.

External validity: In our study, the main concern with external validity lies in the selection of data sources. In order to maximize external validity, we chose to utilize the

“Issues” page of GitHub projects as our data source. GitHub provides an “Issues” page for each open-source project, where practitioners can raise questions, report difficulties, errors, and bugs, and engage in discussions with other practitioners to address these issues. To ensure the extraction of effective information on *causes* and *solutions*, we opted to use closed issue information. Furthermore, GitHub, as the largest open-source project hosting platform globally, offers us sufficient research data to constitute our dataset. However, despite our considerations and the implementation of these strategies, we must admit that we may have overlooked some valuable information.

Reliability: To mitigate potential uncertainties arising from the research methodologies employed, we implemented several measures to enhance the reliability of our study. As mentioned in Section 2.3.1, we conducted pilot studies to assess consistency on data labelling results between the first and third authors before the formal studies. In the pilot data labelling phase, we computed a Cohen’s Kappa coefficient value of 0.838, which indicated good consistency between authors. Throughout the process of data labelling, extraction, and analysis, thorough discussions and resolutions were conducted among the authors to address any internal inconsistencies, ensuring the consistency and accuracy of the results. Furthermore, we have provided the dataset of our study (Cai et al., 2024) for other researchers to replicate this study and validate our findings.

6. Related Work

6.1. Utilization of Pre-Trained Models in Open-Source Projects

Several studies focused on exploring the applications of Pre-Trained Models (PTMs) in open-source projects. Tufano et al. (2024) analyzed 1,501 commits, pull requests, and issues collected from GitHub, identifying instances that suggest the use of ChatGPT in software development. They concluded that practitioners can use ChatGPT to assist in various aspects of software development tasks. Lin et al. (2024) used Latent Dirichlet Allocation (LDA) topic modeling to explore the topics of issues discussed in 200 ChatGPT-related projects, and evaluated the difficulty of resolving various categories of issues based on the average attention and closing rate of an issue category. They also analyzed the temporal trends in the evolution of these issue categories. Pepe et al. (2024) analyzed PTMs from the Hugging Face (HF) platform and identified open-source projects on GitHub utilizing these models. They concluded that it is significant to improve model transparency and documentation to foster trust in AI models. Castaño et al. (2024) collected data from the HF platform and the HFCommunity dataset, then employed neural network methods to classify submission information and conducted LDA topic modeling to analyze model card content. Their study revealed trends in popularity growth of HF and proposed a model maintenance status classification system for models based on multiple attributes to differentiate between high-maintenance and

low-maintenance models. Tan et al. (2024) collected PTM-related questions from the Stack Overflow platform and analyzed the data using statistical and qualitative methods. The research revealed that PTM-related questions have become increasingly popular over time, with low response rates and longer response times. In addition, PTM-related questions were categorized during the analysis. Taraghi et al. (2024) used all public topics from the HF forum and model data from the HF Model Hub to explore the challenges users encounter when utilizing PTMs. Their study revealed new challenges in PTM reuse within the HF community and provided guidance and recommendations for improving PTM reuse for various stakeholders. Tao et al. (2024) conducted an empirical study to analyze challenges faced by LLMs in addressing GitHub issues, identifying factors influencing their performance. They subsequently introduced a novel multi-agent framework called MAGIS, which collaboratively addresses GitHub issues. MAGIS demonstrates significant performance improvements over baseline models and other popular LLMs, achieving an eight-fold improvement in resolution rate compared to the application of GPT-4.

6.2. Exploration of Issues, Causes, and Solutions in Open Source Projects

Several studies used automatic and manual methods to conduct empirical software engineering research on open-source software projects. Yang et al. (2023) selected 576 repositories from the PapersWithCode platform, then collected 24,953 issues from these repositories. They manually analyzed these issues and categorized them. They then examined the resolution status of the issues and explored the use of GitHub issue management features (such as labels and assignees) and their impact on issue resolution. Ultimately, they classified the issues into 13 categories. The two most common issues are runtime errors (23.18%) and unclear instructions (19.53%). Among these, 67.5% of the issues were closed, with half of them resolved within four days. The GitHub issue management features are not widely adopted in open-source AI repositories. Humatova et al. (2020) employed the open coding approach to manually analyze 1,059 artifacts obtained from GitHub issues, pull requests, commits, and related Stack Overflow posts. Additionally, they collected failure experiences encountered by researchers and practitioners during the use of deep learning frameworks through interviews with 20 individuals. Ultimately, they developed a classification system consisting of 5 top-level categories, covering a range of failure types from model architecture to the model training. They validated this classification system through a survey involving 21 developers, confirming that almost all failure categories (13/15) were experienced by at least 50% of the survey participants. Martinez-Torres and Diaz-Fernandez (2014) selected 154 papers published between 2001 and 2011 from the Web of Science database using “open source” and “virtual/online communities” as searching keywords. Subsequently, they performed cluster analysis on the keywords of these papers. Finally, they identified eight distinct research themes, and

discussed the main challenges and academic achievements for each category. Zhao et al. (2023b) collected a total of 570 performance issues from 13 open source projects from the Apache Software Foundation platform. Then, they employed the open coding approach to manually analyze these issues. Finally they summarized eight general types of performance issues with corresponding root causes and resolutions that apply for three languages (i.e., C/C++, Java, Python). They found that only 15% of the performance issues involved revisions to test code, and design-level optimizations typically require greater investment, but do not always lead to higher performance gains. Zahedi et al. (2018) collected the issues from 200 randomly sampled GitHub repositories, and used a mixed-method approach, combining LDA topic modeling and manual thematic analysis, to conduct the study. Finally, they summarized 7 high-level themes of problems that developers face in implementing security features. Waseem et al. (2021) collected 1,345 issue discussions from five microservice systems hosted on GitHub and manually analyzed these issues. Finally they developed a classification system for microservices issues, consisting of 17 categories, 46 subcategories, and 137 issue types, and identified seven categories of root causes for the issues. Waseem et al. (2023) collected data from 2,641 issues from the issue tracking systems of 15 open-source microservice systems on GitHub, conducted 15 interviews, and carried out an online survey completed by 150 practitioners. They used descriptive statistics and constant comparison techniques to analyze the data. Finally, they developed a classification system for microservice issues consisting of 19 categories, 54 subcategories, and 402 types; a classification system for root causes consisting of 8 categories, 26 subcategories, and 228 types; and a classification system for solutions consisting of 8 categories, 32 subcategories, and 177 types.

6.3. Conclusive Summary

As LLMs are increasingly used in our daily life, many open-source software practitioners started to integrate LLMs into open-source software to implement core functionalities (Weber, 2024). In the field of open-source software development, many researchers have already employed automated and manual methods to conduct empirical studies on specific types of open-source projects. However, existing studies did not focus on investigating the issues faced by LLM open-source software from the perspective of practitioners (i.e., developers and users), nor did they explore the underlying causes of these issues and their solutions. Lin et al. (2024) had studied the issues within ChatGPT-related projects, which is a specific and popular commercial LLM, and they utilized the LDA topic modeling, which is an automatic method, to explore the topics of the issues discussed in 200 ChatGPT-related projects, but they did not explore the causes and solutions of these issues. Our study, grounded in the perspective of practitioners, aims to explore the issues within LLM open-source projects, their underlying causes, and potential solutions. We employed a manual qualitative approach to extract and analyze the data collected in the

study. To ensure the representativeness of our research results, we collected projects from GitHub, the largest open-source project hosting platform in the world. Ultimately, Lin et al. (2024) provided a one-tier classification of issues, comprising a total of ten categories, with the top ten keywords for each category identified by the automatic method. However, our taxonomy offers a two-tier classification of issues, comprising a total of 15 categories with corresponding types under each category. We found that only the issue category *Model Reply* in the issue taxonomy identified by Lin et al. (2024) is similar to the issue category *Answer Issue* in our classification. The other categories in their issue taxonomy are quite different from our results.

7. Conclusions

In this study, we focused on the issues that practitioners encounter when developing and using LLM open-source software, as well as their underlying causes and potential solutions. We collected all closed issues from LLM open-source projects on GitHub that met our criteria and labelled a total of 14,476 closed issues that satisfied our requirements. Then we randomly chosen 994 closed issues from the labelled closed issues as the sample data of our study. Finally, we extracted a total of 994 issues, 559 causes, and 798 solutions based on our data extraction criteria. The results show that *Model Issue* is the most common issue faced by practitioners, *Model Problem*, *Configuration and Connection Problem*, and *Feature and Method Problem* are identified as the most frequent causes of the issues, and *Optimize Model* is the predominant solution to the issues.

Based on the study results, we provide implications for practitioners and researchers of LLM open-source projects: Due to the interrelated effects of *Model Issues* and *Parameter Issues* within the software, practitioners should pay more attention to the proper settings of parameters and the validation mechanisms for these parameters. In addition, users can optimize prompts (e.g., constructing prompts according to training data templates) to obtain more satisfactory responses from LLM open-source software. Developers need to be cautious to select suitable models and components when developing LLM open-source software. Besides, researchers should provide automated parameter adjustment tools and methods to ensure better model performance in LLM open-source software. Moreover, researchers should explore potential solutions for reproducing the problems to address those issues that remain unsolved. Researchers can also consider to provide standardized benchmarks to evaluate the impact of different configurations on the performance of models.

Data availability

We have shared the link to our dataset in the reference (Cai et al., 2024).

Acknowledgments

This work is supported by the National Natural Science Foundation of China under Grant No. 62172311 and the Major Science and Technology Project of Hubei Province under Grant No. 2024BAA008.

CRedit authorship contribution statement

Yangxiao Cai: Conceptualization, Investigation, Data curation, Formal analysis, Writing - Original draft preparation. **Peng Liang:** Conceptualization, Methodology, Investigation, Data curation, Supervision, Writing - Original draft preparation. **Yifei Wang:** Investigation, Data curation, Writing - Original draft preparation. **Zengyang Li:** Conceptualization, Methodology, Writing - review and editing. **Mo-jtaba Shahin:** Conceptualization, Methodology, Writing - review and editing.

References

- Cai, Y., Liang, P., Wang, Y., Li, Z., Shahin, M., 2024. Dataset of the Paper “Understanding the Issues, Causes and Solutions in LLM Open-Source Projects”. <https://github.com/Caiyangxiao/LLMOSSIssuesDataset>.
- Castaño, J., Martínez-Fernández, S., Franch, X., Bogner, J., 2024. Analyzing the evolution and maintenance of ml models on hugging face. *arXiv preprint abs/2311.13380*.
- Chase, H., 2024. LangChain. <https://www.langchain.com/langchain>.
- Cohen, J., 1960. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement* 20, 37–46.
- Falkner, S., Klein, A., Hutter, F., 2018. Bohb: Robust and efficient hyperparameter optimization at scale. *arXiv preprint abs/1807.01774*.
- Frantar, E., Alistarh, D., 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot, in: *Proceedings of the 40th International Conference on Machine Learning (ICML)*, ACM. pp. 10323–10337.
- Fu, Y., Liang, P., Tahir, A., Li, Z., Shahin, M., Yu, J., Chen, J., 2025. Security weaknesses of copilot-generated code in github projects: An empirical study. *ACM Transactions on Software Engineering and Methodology*.
- Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J., Wang, H., 2024. Large language models for software engineering: A systematic literature review. *arXiv preprint abs/2308.10620*.
- Humbatova, N., Jahangirova, G., Bavota, G., Riccio, V., Stocco, A., Tonella, P., 2020. Taxonomy of real faults in deep learning systems, in: *Proceedings of the 42nd International Conference on Software Engineering (ICSE)*, ACM. pp. 1110–1121.
- Israel, G.D., 1992. Determining Sample Size. Fact Sheet PEOD-6. Florida, USA.
- Lin, Z., Zhang, N., Liu, C., Zheng, Z., 2024. An empirical study of chatgpt-related projects and their issues on github. *arXiv preprint abs/2403.17437*.
- Liu, Y., Deng, G., Li, Y., Wang, K., Wang, Z., Wang, X., Zhang, T., Liu, Y., Wang, H., Zheng, Y., et al., 2023. Prompt injection attack against llm-integrated applications. *arXiv preprint abs/2306.05499*.
- Malavolta, I., Lewis, G.A., Schmerl, B., Lago, P., Garlan, D., 2021. Mining guidelines for architecting robotics software. *Journal of Systems and Software* 178, 110969.
- Martinez-Torres, M., Diaz-Fernandez, M., 2014. Current issues and research trends on open-source software communities. *Technology Analysis & Strategic Management* 26, 55–68.
- Microsoft, 2024. *semanric-kernel*. <https://github.com/microsoft/semantic-kernel>.
- OpenAI, 2022. ChatGPT. <https://openai.com/chatgpt>.
- Pepe, F., Nardone, V., Mastropaolo, A., Canfora, G., Bavota, G., Di Penta, M., 2024. How do hugging face models document datasets, bias, and licenses? an empirical study, in: *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension (ICPC)*, ACM. pp. 370–381.
- Stol, K.J., Ralph, P., Fitzgerald, B., 2016. Grounded theory in software engineering research: a critical review and guidelines, in: *Proceedings of the 38th International Conference on Software Engineering (ICSE)*, ACM. pp. 120–131.
- Tan, X., Li, T., Chen, R., Liu, F., Zhang, L., 2024. Challenges of using pre-trained models: the practitioners’ perspective, in: *Proceedings of the 31st IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, IEEE. pp. 67–78.
- Tao, W., Zhou, Y., Zhang, W., Cheng, Y., 2024. Magis: Llm-based multi-agent framework for github issue resolution. *arXiv preprint abs/2403.17927*.
- Taraghi, M., Dorcelus, G., Foundjem, A., Tambon, F., Khomh, F., 2024. Deep learning model reuse in the huggingface community: Challenges, benefit and trends. *arXiv preprint abs/2401.13177*.
- Tufano, R., Mastropaolo, A., Pepe, F., Dabić, O., Penta, M.D., Bavota, G., 2024. Unveiling chatgpt’s usage in open source projects: A mining-based study. *arXiv preprint abs/2402.16480*.
- Waseem, M., Liang, P., Ahmad, A., Khan, A.A., Shahin, M., Abrahamsson, P., Nasab, A.R., Mikkonen, T., 2023. Understanding the issues, their causes and solutions in microservices systems: An empirical study. *arXiv preprint abs/2302.01894*.
- Waseem, M., Liang, P., Shahin, M., Ahmad, A., Nassab, A.R., 2021. On the nature of issues in five open source microservices systems: An empirical study, in: *Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, ACM. pp. 201–210.
- Weber, I., 2024. Large language models as software components: A taxonomy for llm-integrated applications. *arXiv preprint abs/2406.10300*.
- Yang, Z., Wang, C., Shi, J., Hoang, T., Kochhar, P., Lu, Q., Xing, Z., Lo, D., 2023. What do users ask in open-source ai repositories? an empirical study of github issues, in: *Proceedings of the 20th International Conference on Mining Software Repositories (MSR)*, IEEE. pp. 79–91.
- Zahedi, M., Babar, M.A., Treude, C., 2018. An empirical study of security issues posted in open source projects, in: *Proceedings of the 51st International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, IEEE. pp. 5504–5513.
- Zhao, W.X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., Liu, P., Nie, J.Y., Wen, J.R., 2023a. A survey of large language models. *arXiv preprint abs/2303.18223*.
- Zhao, Y., Xiao, L., Bondi, A.B., Chen, B., Liu, Y., 2023b. A large-scale empirical study of real-life performance issues in open source projects. *IEEE Transactions on Software Engineering* 49, 924–946.
- Zheng, D., Wang, Y., Shi, E., Zhang, R., Ma, Y., Zhang, H., Zheng, Z., 2024. Towards more realistic evaluation of llm-based code generation: an experimental study and beyond. *arXiv preprint abs/2406.06918*.
- Zheng, Z., Ning, K., Chen, J., Wang, Y., Chen, W., Guo, L., Wang, W., 2023. Towards an understanding of large language models in software engineering tasks. *arXiv preprint abs/2308.11396*.
- Zhou, X., He, J., Ke, Y., Zhu, G., Gutiérrez-Basulto, V., Pan, J.Z., 2024. An empirical study on parameter-efficient fine-tuning for multimodal large language models, in: *Proceedings of the 62nd the Association for Computational Linguistics (ACL)*, ACL. pp. 10057–10084.