Understanding Large Language Model Performance in Software Engineering: A Large-scale Question Answering Benchmark

Ruida Hu*
200111107@stu.hit.edu.cn
Haribin Institute of Technology, Shenzhen
Shenzhen, China

Chao Peng
pengchao.x@bytedance.com
ByteDance
Beijing, China

Jingyi Ren
jingyi.422@bytedance.com
ByteDance
Shenzhen, China

Bo Jiang
jiangbo.jacob@bytedance.com
ByteDance
Shenzhen, China

Xiangxin Meng
mengxiangxin.1219@bytedance.com
ByteDance
Beijing, China

Qinyun Wu
wuqinyun@bytedance.com
ByteDance
Beijing, China

Pengfei Gao
gaopengfei.se@bytedance.com
ByteDance
Beijing, China

Xinchen Wang*
200111115@stu.hit.edu.cn
Haribin Institute of Technology, Shenzhen
Shenzhen, China

Cuiyun Gao†
gaocuiyun@hit.edu.cn
Haribin Institute of Technology, Shenzhen
Shenzhen, China

Abstract

In this work, we introduce **CodeRepoQA**, a large-scale benchmark specifically designed for evaluating repository-level question-answering capabilities in the field of software engineering. CodeRepoQA encompasses five programming languages and covers a wide range of scenarios, enabling comprehensive evaluation of language models. To construct this dataset, we crawl data from 30 well-known repositories in GitHub, the largest platform for hosting and collaborating on code, and carefully filter the raw data. In total, CodeRepoQA is a multi-turn question-answering benchmark with 585,687 entries. It covers a diverse array of software engineering scenarios, with an average of 6.62 dialogue turns per entry.

We evaluate ten popular large language models on our dataset and provide in-depth analysis. We find that LLMs still have limitations in question-answering capabilities in the field of software engineering, and medium-length contexts are more conducive to their performance. The entire benchmark and details are publicly available at https://github.com/kinesiatricssxilm14/CodeRepoQA.

CCS Concepts

• **Software and its engineering → Software libraries and repositories**.

* Work done during an internship at ByteDance.
† Corresponding author.

Keywords

Question Answering, Language Model, Mining Software Repository

ACM Reference Format:
Ruida Hu*, Chao Peng, Jingyi Ren, Bo Jiang, Xiangxin Meng, Qinyun Wu, Pengfei Gao, Xinchen Wang*, and Cuiyun Gao†. 2025. Understanding Large Language Model Performance in Software Engineering: A Large-scale Question Answering Benchmark. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '25), July 13–18, 2025, Padua, Italy.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3726302.3730262

1 Introduction

Large language models (LLMs) are increasingly being integrated into tools such as chatbots and coding assistants to assist developers, showcasing their potential to solving various software engineering tasks [7, 10]. As a result, the research community has begun exploring how LLMs can be further leveraged to assist with more complex repository-level tasks encountered in software development [1, 11, 12, 14, 15]. GitHub issues provide rich, real-world data that includes bug reports, feature requests, and usage questions, making them an ideal source for constructing diverse and realistic question-answering tasks for evaluating LLMs. Their collaborative nature and technical content allow models to be assessed on practical software development challenges.

Existing question-answering (QA) benchmarks, such as MMLU [6], evaluate models in zero-shot and few-shot settings and assess them in answering question across 57 subjects spanning science, technology and mathematics. CodeQA [9] contains a benchmark of 119,778 Java and 70,085 Python QA pairs. CS1QA [8] contains 9,237 question-answer pairs extracted from chat logs of introductory Python classes. CodeApex [2] evaluates LLMs on C++ code generation and correction.

Additionally, these benchmarks do not focus on repository-level question-answering and cannot reflect the complexity of real-world

Table 1: The comparison between existing benchmarks and CodeRepoQA.

Benchmark	#Langs	#Samples	Multi-turn	Repo-level
MMLU [6]	-	15,908	✗	✗
CodeQA [9]	2	190,000	✗	✗
CS1QA [8]	1	9,237	✗	✗
CodeApex [2]	1	250	✗	✗
CodeRepoQA	**5**	**585,687**	✓	✓

Table 2: Statistics of each programming language in CodeRepoQA, where #ave-turn indicates average turns of dialogues.

Languages	Python	Java	Typescript	Javascript	Go	All
#entries	204,501	34,792	220,884	39,246	86,264	585,687
#ave-turn	6.61	4.31	5.89	6.98	9.32	6.62

scenarios. Specifically, developers often engage in multi-turn dialogues to resolve issues. Multi-turn dialogues better reflect real-world conversations, as solving software issues often requires multiple interactions [3]. These benchmarks are limited to single-turn dialogues, which fail to capture the multi-turn interactions needed to resolve software engineering issues. The comparison between CodeRepoQA and other benchmarks can be found in Table 1.

To address the above challenges in existing benchmarks, we present CodeRepoQA, a novel large-scale benchmark derived from conversations in GitHub repository issues. The benchmark contains 585,687 entries, with an average of 6.62 dialogue turns per conversation. CodeRepoQA encompasses five widely used programming languages, with data details provided in Table 2.

We also conduct experiments with state-of-the-art models in the CodeRepoQA, such as GPT, DeepSeek-Coder, and assess the accuracy of the model predictions against ground truth, using metrics such as BLEU, ROUGE-L, ROUGE-1, and Edit Similarity. We find that LLMs still demonstrate limitations in real-world QA scenarios, and medium-length contexts are more conducive to LLMs' performance. In summary, the paper makes the following contributions:

(1) **Large-scale repository-level QA benchmark from GitHub.** We build a large-scale QA benchmark derived from real-world GitHub repositories. This benchmark includes 585,687 QA entries in five programming languages and encompasses a wide variety of tasks that reflect the complexities of actual software development and maintenance, offering a more realistic assessment of LLM capabilities.
(2) **Multi-turn software engineering dialogue.** Our benchmark includes multi-turn dialogues, derived from real developer feedback during the development process, with an average of 6.62 dialogue turns per entry.
(3) **Comprehensive experimental results.** We evaluate state-of-the-art LLMs on the QA scenario. Our experiments reveal key insights into the effectiveness of these models, highlighting their limitations in the QA capabilities within the field of software engineering and the fact that medium-length contexts are more conducive to LLMs' performance.

Table 3: Basic information for each entry in GitHub issues, including Base Information, Content Information.

Features	Description
Base Information	
language	The primary programming language used in the repository.
type	Whether the entry is an Issue or a PR.
repo	The repository's name.
owner	The repository owner, either an individual or an organization.
number	The unique identifier for the issue or PR.
author-info	Details about the author of the issue or PR.
author-association	The author's relationship with the repository (e.g., member).
state	The current status of the issue or PR (e.g., open, closed).
created-at	The timestamp when the issue or PR was created.
updated-at	The timestamp of the most recent update to the issue or PR.
Content Information	
title	The title of the issue or PR.
body	The main content or description of the issue or PR.
comments	The content of comments on the issue or PR.

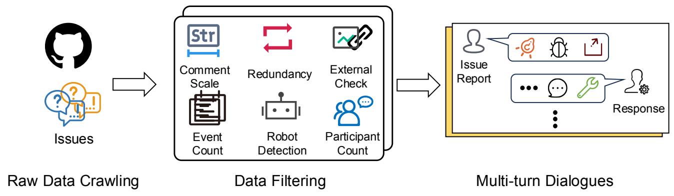


Figure 1: The process of data construction.

2 Data Construction

As illustrated in Figure 1, we employ two steps to produce the final benchmark: **(1) Raw Data Crawling**: we crawl and extract metadata related to issues from GitHub[1]; **(2) Data Filtering**: we select and clean the gathered data to ensure its relevance and quality. After completing the above two steps, we obtain the final benchmark with diverse issue attributes. As shown in Figure 2, the data includes key attributes such as repository information (owner, repo, time) and main QA details (issue title, issue body, and responses). Additionally, the role indicates the user's relationship to the repository, such as contributor or member.

2.1 Raw Data Crawling

To ensure sufficient diversity within CodeRepoQA, we select 30 popular repositories from GitHub, encompassing five widely-used programming languages: Python, Java, JavaScript, TypeScript, and Go. To assure the quality of the benchmark, we select repositories with over 5,000 stars that are widely utilized within their respective fields. GitHub, as the largest platform for hosting and managing software projects, contains a wealth of metadata. The GitHub REST API [4] allows developers to interact with GitHub services through HTTP requests. We use the GitHub REST API to crawl all issues from 30 repositories on GitHub, totalling over 636,000 entries. To ensure the quality of CodeRepoQA, we conducted the following filtering. The basic information for each entry is shown in Table 3, including base information and content information.

[1]The crawling was performed in August 2024, and the complete structure is displayed in our GitHub repository

2.2 Data Filtering

After raw data crawling, we obtain approximately 636,000 issues. These issues are intended for constructing the QA benchmark. We apply the following criteria to filter the issue entries:

Comment scale: We prioritize character count over the number of comments. Entries under 200 characters are removed for lack of content, while those exceeding 10MB or with very high character counts are also excluded to avoid overwhelming the model's capacity. This balances providing sufficient information with staying within the model's input limits.

Redundancy: GitHub hosts many duplicate issues, which generally contain repetitive content and lack novelty, making them unsuitable as evaluation datasets. According to GitHub documentation [5], an issue is marked as a duplicate by GitHub if its comments contain "`Duplicate of #`" followed by an issue or PR number. We detect these markers and related keywords to exclude redundant issues, ensuring each selected issue is unique and valuable for evaluation, thereby enhancing data quality.

External check: In reviewing issues, we often encounter a variety of external links, including web pages, images, and videos. Due to the diversity and management challenges of these links, we opt to retain only those issues containing internal GitHub links. This criterion helps avoid the complications of analyzing diverse external content, ensuring our dataset remains focused and manageable, thus improving our data processing and analysis efficiency.

Event count: Influenced by methodologies from research like StarCoder[13], we consider the impact of event numbers within an issue. Issues with more than ten events typically contain a large amount of auto-generated text and robot-produced records, such as extensive logs and unnecessary links. These elements lower data quality and increase processing complexity. Therefore, we exclude issues with more than ten events to maintain a dataset of focused and high-quality entries, enabling more effective analyses.

Robot detection: We implement measures to eliminate robot influences. The initial step involved reviewing the "`author-info`" field to identify robot users by searching for terms like "bot" in user types and names. Once identified, all issues and responses created by these robots are removed. We also catalog common robot response patterns to ensure the removal of these automated interactions from the dataset.

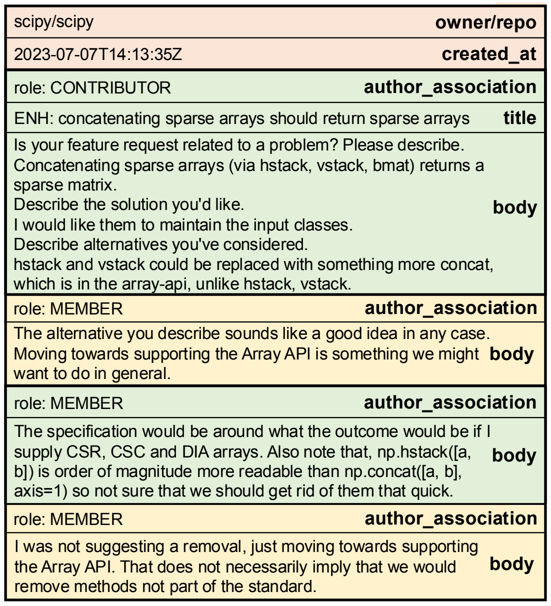
Participant count: We evaluate the number of participants per issue to ensure discussions in our dataset are lively and diverse. After removing robot users, we exclude issues with only one participant, which often indicate unresolved problems. This approach helps ensure our dataset contains genuine and effective interactions.

After completing the data filtering process, we obtained 585,687 issue entries containing numerous real-world code-related QA pairs. In conclusion, our benchmark includes five programming languages; the number of entries and dialogue turns for each language are detailed in Table 2.

3 Experiment

The experiments use filtered QA pairs as the benchmark. For our constructed benchmark, CodeRepoQA, we propose the following two Research Questions (RQs) based on the specified aspects:

RQ1: How do models perform in answering questions?



scipy/scipy	**owner/repo**
2023-07-07T14:13:35Z	**created_at**
role: CONTRIBUTOR	**author_association**
ENH: concatenating sparse arrays should return sparse arrays	**title**
Is your feature request related to a problem? Please describe. Concatenating sparse arrays (via hstack, vstack, bmat) returns a sparse matrix. Describe the solution you'd like. I would like them to maintain the input classes. Describe alternatives you've considered. hstack and vstack could be replaced with something more concat, which is in the array-api, unlike hstack, vstack.	**body**
role: MEMBER	**author_association**
The alternative you describe sounds like a good idea in any case. Moving towards supporting the Array API is something we might want to do in general.	**body**
role: MEMBER	**author_association**
The specification would be around what the outcome would be if I supply CSR, CSC and DIA arrays. Also note that, np.hstack([a, b]) is order of magnitude more readable than np.concat([a, b], axis=1) so not sure that we should get rid of them that quick.	**body**
role: MEMBER	**author_association**
I was not suggesting a removal, just moving towards supporting the Array API. That does not necessarily imply that we would remove methods not part of the standard.	**body**

Figure 2: A multi-turn QA entry illustrating the main components of the entry in CodeRepoQA.

RQ2: How does the length of question affect the models' answering performance?

3.1 Model Selection

We select ten different large language models (LLMs) to evaluate their performance on coding tasks. These models are broadly used and represent some of the most advanced technologies available, including both commercial and open-source options. Commercial models include the GPT and Gemini (GM) series, such as GPT-4o, GPT-4, Gemini-1.5-Flash, and Gemini-1.5-Pro. Open-source models include Mistral, CodeQwen (CQ), and the DeepSeek Coder (DSC) family, encompassing a range of sizes. These include Mistral-large-2 (123B), CodeQwen-1.5-Chat (7B), DeepSeek-Coder-V2 (236B), DeepSeek-Coder-V2-Lite (16B), DeepSeek-Coder (33B), and DeepSeek-Coder (6.7B). To ensure the validity of all experiments, we control the length of the input data to ensure it did not exceed the maximum input tokens allowed by the selected models.

3.2 Evaluation Design

To comprehensively assess the capability of LLMs in question-answering in software engineering scenarios, we designed a specific QA task. For this evaluation, we use the historical dialogue turns as input, ensuring that the model understands the context of the conversation. The last response from a repository maintainer (such as MEMBER, AUTHOR, or CONTRIBUTOR) is used as the ground truth. This approach allows us to accurately answer the LLMs' ability to answer the questions in software engineering scenarios.

3.3 Evaluation Metrics

We employ the following metrics to evaluate LLMs:

BLEU: It measures the n-gram precision between the generated text and a set of reference texts, incorporating a brevity penalty to

Table 4: Performance of various LLMs on CodeRepoQA. In the table, color shades of each block denote performance rankings: darkest for highest, medium for second highest, and lightest for third highest scores.

Metric	GPT-4o	GPT-4	GM-Flash	GM-Pro	Mistral-123B	CQ-7B	DSC-236B	DSC-16B	DSC-33B	DSC-6.7B	Average
BLEU	0.0943	0.1179	0.1227	0.1208	0.0948	0.1188	0.1102	0.0942	0.1184	0.1182	0.1110
ROUGE-L	0.1189	0.1330	0.1499	0.1551	0.1228	0.1392	0.1340	0.1201	0.1401	0.1366	0.1350
ROUGE-1	0.1984	0.2315	0.2410	0.2470	0.1988	0.2264	0.2203	0.1961	0.2272	0.2279	0.2215
Edit Similarity	0.1388	0.1715	0.1914	0.2074	0.1411	0.1803	0.1615	0.1406	0.1785	0.1784	0.1689
Average	0.1376	0.1635	0.1762	0.1826	0.1393	0.1662	0.1565	0.1377	0.1660	0.1653	0.1591

Table 5: The impact of different context lengths. The dark blocks indicate the group with the best performance for the metric. The ~ symbol followed by a percentage indicates the proportion of each group's length.

Metric	~20%	~40%	~60%	~80%	~100%	Average
BLEU	0.1077	0.1196	0.1107	0.1092	0.1080	0.1110
ROUGE-L	0.1393	0.1440	0.1373	0.1312	0.1230	0.1350
ROUGE-1	0.2240	0.2348	0.2212	0.2132	0.2140	0.2215
Edit Similarity	0.1708	0.1806	0.1664	0.1648	0.1621	0.1689
Average	0.1605	0.1698	0.1589	0.1546	0.1518	0.1591

discourage overly concise outputs. This metric is primarily used to assess the linguistic accuracy of the generated text.

ROUGE-L and ROUGE-1: These metrics assess the quality of text generated by models compared to reference texts. ROUGE-L evaluates sequence-level similarity using the Longest Common Subsequence (LCS) method, while ROUGE-1 measures the overlap of single words (1-grams). The F1 score is the primary metric for a balanced evaluation of content accuracy and completeness.

Edit Similarity: It evaluates the similarity between the generated text and the reference text by comparing the number of edits required to transform one text into the other.

3.4 Performance of LLMs

We evaluate the performance of ten models listed in Section 3.1. The results for the evaluation are presented in the respective sections of Table 4. We can derive the following observations:

Commercial models do not always outperform open-source ones in software engineering, nor do larger models consistently show superior performance. Although two Gemini models achieve the best performance, GPT-4, a commercial model, underperforms compared to open-source models like CodeQwen (7B). Additionally, the largest and newest model in the DeepSeek series, DeepSeek-Coder-V2, is outperformed by its smaller counterparts, 33B and 6.7B. This suggests that having more parameters does not necessarily lead to better performance, highlighting that larger models are not always superior for different tasks.

LLMs still demonstrate limitations in software engineering QA scenarios. Even with Gemini-1.5-Pro, which has the best overall performance, there is a significant gap between its output and the actual feedback. The combined score across four metrics is only 0.1826, with a BLEU score of 0.1208. Specifically, Gemini-1.5-Pro achieved the highest scores in ROUGE-L (0.1551) and ROUGE-1 (0.2470). This indicates that the vocabulary overlap between the generated text and real-world responses is relatively low, suggesting that the generated content may not effectively capture key information from the actual responses.

3.5 Impact of Question Length

To investigate the impact of question length on performance of LLMs, we equally divide the benchmark into five groups based on the question length. We then calculate the scores for each group across four metrics. The detailed results are shown in Table 5.

LLMs perform better on questions of medium length in software engineering QA scenarios. Based on the data in Table 5, it is evident that LLMs perform better on questions of medium length. The 40% context length group achieves the highest scores in BLEU (0.1196), ROUGE-1 (0.2348), and Edit Similarity (0.1806), and maintains the highest average score (0.1698) among all groups. Similarly, the 60% context length group also performs well, particularly in BLEU, ROUGE-L, and ROUGE-1 metrics, with an average score of 0.1589. In contrast, the shortest (20%) and longest (100%) context length groups show relatively lower performance across most metrics, indicating that extremely short and long contexts are less effective. These findings suggest that medium-length contexts are more conducive to generating high-quality answers from LLMs in software engineering QA scenarios.

4 Conclusion

We present CodeRepoQA, a large-scale benchmark for assessing the question-answering capabilities of LLMs in the field of software engineering. It is a multi-turn question-answering benchmark, containing 585,687 entries from 30 well-known GitHub repositories and covers five programming languages. CodeRepoQA provides a comprehensive evaluation of LLMs' abilities in answering questions of repository level compared to previous benchmarks. Our experiments show that LLMs still demonstrate limitations in software engineering QA scenarios, and medium-length questions are found to yield better performance in generating high-quality answers.

5 Acknowledgement

This research is supported by National Key R&D Program of China (No. 2022YFB3103900), National Natural Science Foundation of China under project (No. 62472126), Natural Science Foundation of Guangdong Province (Project No. 2023A1515011959), Shenzhen International Science and Technology Cooperation Project (No. GJHZ20220913143008015), Shenzhen-Hong Kong Jointly Funded Project (Category A, No. SGDX20230116091246007), and Shenzhen Basic Research (General Project No. JCYJ20220531095214031).

References

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

[2] Lingyue Fu, Huacan Chai, Shuang Luo, Kounianhua Du, Weiming Zhang, Longteng Fan, Jiayi Lei, Renting Rui, Jianghao Lin, Yuchen Fang, et al. 2023. Codeapex: A bilingual programming evaluation benchmark for large language models. *arXiv preprint arXiv:2309.01940* (2023).

[3] Xingyuan Bu Ge Bai, Jie Liu et al. 2024. MT-Bench-101: A Fine-Grained Benchmark for Evaluating Large Language Models in Multi-Turn Dialogues. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*. ACL, 7421–7454.

[4] github. 2022. GitHub REST API documentation. https://docs.github.com/en/rest?apiVersion=2022-11-28.

[5] github. [n.d.]. marking-issues-or-pull-requests-as-a-duplicate. https://docs.github.com/en/issues/tracking-your-work-with-issues/administering-issues.

[6] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring Massive Multitask Language Understanding. *Proceedings of the International Conference on Learning Representations (ICLR)* (2021).

[7] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2023. Large language models for software engineering: A systematic literature review. *arXiv preprint arXiv:2308.10620* (2023).

[8] Changyoon Lee, Yeon Seonwoo, and Alice Oh. 2022. CS1QA: A Dataset for Assisting Code-based Question Answering in an Introductory Programming Course. *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (2022).

[9] Chenxiao Liu and Xiaojun Wan. 2021. CodeQA: A question answering dataset for source code comprehension. *arXiv preprint arXiv:2109.08365* (2021).

[10] Junwei Liu, Kaixin Wang, Yixuan Chen, Xin Peng, Zhenpeng Chen, Lingming Zhang, and Yiling Lou. 2024. Large Language Model-Based Agents for Software Engineering: A Survey. *arXiv preprint arXiv:2409.02977* (2024).

[11] Yizhou Liu, Pengfei Gao, Xinchen Wang, Chao Peng, and Zhao Zhang. 2024. MarsCode Agent: AI-native Automated Bug Fixing. *arXiv preprint arXiv:2409.00899* (2024).

[12] Stephen MacNeil, Andrew Tran, Arto Hellas, Joanne Kim, Sami Sarsa, Paul Denny, Seth Bernstein, and Juho Leinonen. 2023. Experiences from using code explanations generated by large language models in a web software development e-book. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. 931–937.

[13] Yangtian Zi Raymond Li, Loubna Ben Allal et al. 2023. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161* (2023).

[14] Steven I Ross, Fernando Martinez, Stephanie Houde, Michael Muller, and Justin D Weisz. 2023. The programmer's assistant: Conversational interaction with a large language model for software development. In *Proceedings of the 28th International Conference on Intelligent User Interfaces*. 491–514.

[15] John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. Swe-agent: Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv:2405.15793* (2024).