DCA-Bench: A Benchmark for Dataset Curation Agents

Benhao Huang[1] **Yingzhuo Yu**[3] **Jin Huang**[2] **Xingjian Zhang**[2] **Jiaqi Ma**[3]
[1] Shanghai Jiao Tong University [2] University of Michigan
[3] University of Illinois Urbana-Champaign

Abstract

The quality of datasets plays an increasingly crucial role in the research and development of modern artificial intelligence (AI). Despite the proliferation of open dataset platforms nowadays, data quality issues, such as insufficient documentation, inaccurate annotations, and ethical concerns, remain common in datasets widely used in AI. Furthermore, these issues are often subtle and difficult to be detected by rule-based scripts, requiring expensive manual identification and verification by dataset users or maintainers. With the increasing capability of large language models (LLMs), it is promising to streamline the curation of datasets with LLM agents. In this work, as the initial step towards this goal, we propose a dataset curation agent benchmark, DCA-Bench, to measure LLM agents' capability of detecting hidden dataset quality issues. Specifically, we collect diverse real-world dataset quality issues from eight open dataset platforms as a testbed. Additionally, to establish an automatic pipeline for evaluating the success of LLM agents, which requires a nuanced understanding of the agent outputs, we implement a dedicated Evaluator using another LLM agent. We demonstrate that the LLM-based Evaluator empirically aligns well with human evaluation, allowing reliable automatic evaluation on the proposed benchmark. We further conduct experiments on several baseline LLM agents on the proposed benchmark and demonstrate the complexity of the task, indicating that applying LLMs to real-world dataset curation still requires further in-depth exploration and innovation. Finally, the proposed benchmark can also serve as a testbed for measuring the capability of LLMs in problem discovery rather than just problem-solving. The benchmark suite is available at https://github.com/TRAIS-Lab/dca-bench.

1 Introduction

High-quality datasets have become increasingly crucial for advancing artificial intelligence (AI) [1, 2]. Open datasets platforms, such as Hugging Face [3] or BIG-Bench [4], have substantially accelerated AI research and development by facilitating community contributions. However, community-contributed datasets often encounter subtle data quality issues, including insufficient documentation [5], inaccurate annotations [6], ethical concerns [7], and inconsistency across multiple files (see Appendix A.1 for real-world examples).

There have been existing efforts on standardizing dataset management practices [7–9], and developing dataset curation toolkits and systems [10, 11]. However, these toolkits and systems often rely heavily on rule-based scripts, which lack the necessary flexibility to detect the aforementioned subtle data quality issues. Consequently, existing techniques are insufficient to tackle the complex challenge of automatically curating datasets at scale on open dataset platforms, where we still primarily rely on dataset users or platform maintainers to identify data quality issues.

Preprint. Under review.

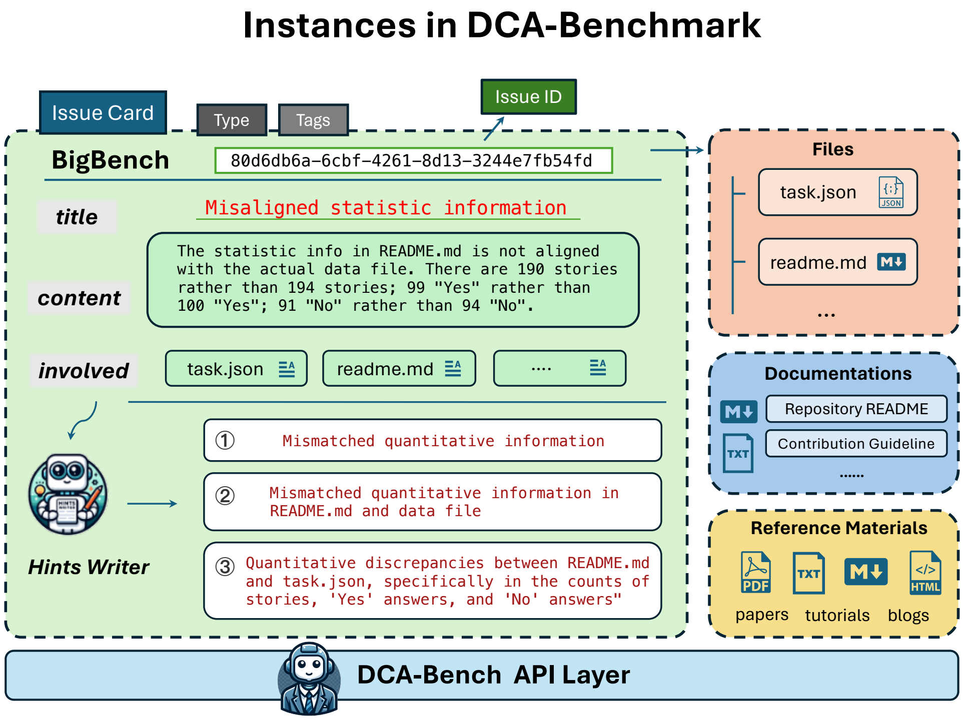

Figure 1: Illustration of the instances in DCA-Bench. The Issue Card displays a specific test case in DCA-Bench. Each test case includes essential metadata such as ID, source platform (BIG-Bench in this case), type, tags, hints generated, and other items, as shown in the Issue Card. Relevant dataset files can be found using the Issue ID. Furthermore, DCA-Bench incorporates documentation from the dataset platform along with additional reference materials related to dataset curation and quality. These components are integrated into the API Layer for convenient access. The Curator is asked to detect the issues in files by describing the issue context and pinpointing the location in the file where issues occur. The elements labeled as "title", "content", and "involved" are considered as the ground truth, which will be used for evaluating the Curator and is not available as inputs to the Curator.

Recent advancement of large language models (LLMs) has led to promising development of LLM agents for real-world software engineering problems [12]. For certain issues in software development, LLM agents have been shown to be capable of autonomously generating proper code to fix them [13, 14]. Given this development, we envision that LLM agents could become an effective technique for building autonomous dataset curation systems.

However, there is still a significant gap between the need of dataset curation and the state-of-the-art LLM agents for software engineering. Specifically, existing studies primarily focus on developing agents that solve identified and well-defined issues, while dataset curation requires one to discover hidden issues in the community-contributed datasets—a distinct capability different from problem-solving [15, 16] and remaining under-explored for LLM agents.

In response to these challenges, we introduce the Dataset Curation Agents Benchmark (DCA-Bench) as an initial step towards achieving autonomous dataset curation. Specifically, DCA-Bench aims to provide a comprehensive benchmark for evaluating LLM agents' capability to discover data quality issues across online dataset platforms, the first step of the curation pipeline. Henceforth, we will consistently refer to such an LLM agent as a **"Curator"** to highlight its role in this task. A well-performed Curator can detect and locate existing issues, which is critical for a follow-up fix by human maintainers or other LLM agents. We collected 91 representative samples from 8 online dataset platforms and classified them into 4 types with 18 tags according to their various content and difficulty. The main features of DCA-Bench include:

- **Real-world Cases with Minimal Simplification**: All test cases of DCA-Bench have references to real-world sources, allowing benchmark users to better understand them in the wild. To test the Curator's ability under the complex real-world environment, DCA-Bench incorporates all relevant dataset files for each test case, not only limited to those flawed ones.

- **Multiple Difficulty Levels**: DCA-Bench provides four levels of hints for each test case in the benchmark. From a higher level hint, the Curator gains more information about the content and location of the issue. The motivation is to make the task more achievable and also gauge the information required for the Curator to detect these issues.
- **Accurate Automatic Evaluation** Unlike traditional machine learning tasks, the task of dataset curation does not have labels that can be directly evaluated by scripts. Human-level efforts are required to rate the performance of the Curator, which is not scalable. Therefore, we develop an automatic and accurate evaluation scheme using GPT4 [17] to replace human annotators.

In addition to the benchmark, we implement a straightforward baseline for this task using OpenAI GPT4 Assistant[1]. The provided baseline only succeeds in detecting 10.99% issues without hints and 62.64% when given the most specific hint (see Section 3.2), demonstrating the difficulty of this benchmark.

We believe DCA-Bench will serve as a foundational initial step towards developing a fully autonomous and powerful dataset curation system, further enhancing the quality of community-contributed open datasets. This benchmark can also serve as a testbed for evaluating LLMs' capability of problem discovery in addition to problem-solving, which is a critical area that has been under-explored.

We organize this paper as follows: In Section 2, we introduce the background of the dataset curation problem and existing literature. In Section 3, we go through an overview of the composition of DCA-Bench, the construction process, task definition as well as how we evaluate the performance of the Curator. In Section 4, we carry out experiments to validate the reliability of our automatic evaluation scheme and show preliminary testing results of baseline Curators on DCA-Bench.

2 Related Work

Dataset Quality Management With the advancement of modern AI, the need for high-quality datasets is surging [1, 2]. Consequently, open dataset platforms like Hugging Face and Kaggle keep growing rapidly. However, recent studies [6, 18, 19] indicate that many popular datasets suffer from errors, biases, or annotation artifacts [20], and lack proper documentation [5]. In addition, common issues in real-world datasets include problems with data loading scripts and file discrepancies, where conflicting information exists between files.

There have been many exisiting efforts on dataset quality management. Gebru et al. [7] outline principles for creating and managing datasets, especially in the context of machine learning. Gong et al. [21] propose several strategies, including dataset profiling, data cleansing, and quality monitoring. More recent works [7–9] further explore standardizing dataset management practices. Based on these works, researchers have developed several dataset curation toolkits and systems [10, 11]. More recently, Zhou et al. [22] developed a system that is relevant to enhancing data quality. However, they focus on filtering a large-scale LLM-pertaining corpus rather than detecting dataset quality issues. Besides, there are some specialized works focused on detecting harmful content [23, 24] and annotation errors [19, 25]. Nevertheless, those methods are not generalizable enough and lack flexibility to solve challenges such as nuanced ethical biases (see Tab. 8) and incorrect annotations involving logical or factual errors (see Tab. 10). Moreover, studies on file discrepancies(see Tab. 9) and issues remain scarce. Consequently, we still primarily rely on dataset users or platform maintainers to identify these issues in practice.

We envision LLM agents as a promising technique to address the challenges in this space, and propose a benchmark dataset as an initial step towards this goal.

LLMs for Software Engineering Our work is also relevant to the emerging research direction on using LLMs to solve software engineering tasks. Fan et al. [26] suggest that there are two major categories in this area: (i) code generation and completion, and (ii) maintenance and evolution.

There has been extensive literature on LLMs for code generation and completion, in terms of both models and benchmarks. On the model side, there have been various code generation LLMs [27–29].

[1]We use `GPT-4-0125-preview` with Code Interpreter enabled.

On the benchmark side, datasets have been created to complete functions and short programs from natural language descriptions [12, 30–34] and retrieve relevant code [35]. We note that the main purpose of the proposed DCA-Bench is to measure LLM agents' capability of discovering hidden dataset issues instead of generating code, although generating and running some test code may be helpful.

For the second category, maintenance and evolution, it involves tasks such as localizing and fixing bugs [36, 37], improving program performance [38, 39], and refactoring code without changing program behavior [40]. Our work falls into this category while differing from previous studies in two key aspects. Firstly, while most literature focuses on tasks with well-defined outcomes, such as fixing a buggy function, our work addresses non-standard issues like ethical concerns in data points or cross-file inconsistencies. Secondly, few studies have explored using LLMs to maintain dataset repositories, which is increasingly important for the machine learning community.

LLMs as A Proxy of Human Evaluation Recent studies have shown that LLMs, when carefully prompted, can serve as a good proxy of human evaluations for a number of scenarios, such as evaluating text generation quality [41, 42], reasoning ability [43, 44], and generated image quality [45, 46].

Inspired by these studies, we adopt an LLM to automatically evaluate the Curator's responses based on carefully designed instruction. We also empirically show that our LLM-based Evaluator highly aligns with human preference, ensuring a reliable automatic evaluation scheme.

3 The Dataset Curation Agent Benchmark

In this section, we start with an overview of the proposed DCA-Bench in Section 3.1. Following that, we introduce the benchmark test case construction process in Section 3.2 and give a detailed task definition in Section 3.3. In Section 3.4, we introduce our evaluation pipeline in DCA-Bench.

3.1 Benchmark Overview

At a high level, DCA-Bench consists of three main parts: **benchmark assets**, a **benchmark API**, and an **evaluation pipeline**.

Benchmark assets include *test cases* and *reference resources*. For test cases, DCA-Bench includes 91 dataset issues from eight dataset platforms. Statistics about the benchmark are shown in Tab. 1. Each test case includes multiple files, providing a minimal environment for identifying hidden issues. We classify those cases according to the number of files and issues involved, and further assign 18 tags to categorize them, which include "data-problem", "document-problem", "infrastructure-problem", "ethical/legal-risk", and more. Statistics by file and issue count are shown in Tab. 2. The detailed definitions of the types and tags are provided in Appendix A.3. Readers can refer to Appendix A.10 for some concrete examples of dataset issues.

For reference resources, DCA-Bench includes each platform's documentation and external materials. Depending on each platform, the documentation may consist of the README files, contribution guidelines, and dataset formats, setting out specific rules for contributing to the platform. The external materials encompass research papers on dataset curation, blogs about best practices for dataset maintenance, and various tutorials. These resources offer additional context that might help identify discrepancies between platform requirements and the actual datasets. Benchmark users can utilize this information to improve the performance of their Curators using RAG [47] or other methods.

The **benchmark API** is a Python library associated with the DCA-Bench, where it loads the benchmark assets and provides formatted inputs to the Curators and the LLM-based Evaluator. Benchmark users can use the API to easily run their Curator on the benchmark.

The key component of the **evaluation pipeline** is an LLM-Based *Evaluator*, which is an LLM agent that automatically evaluates the performance of the Curators. The detailed design of the Evaluator is discussed in Section 3.4, and we demonstrate that it empirically aligns well with human evaluations in Section 4.1.

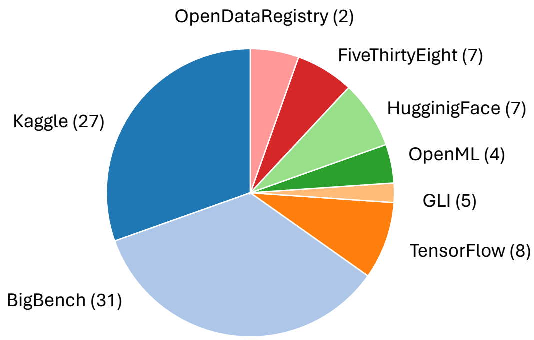

Figure 2: Distribution of source platform in DCA-Bench.

Table 1: File statistics of DCA-Bench. #Token is calculated using the `tiktoken` package to tokenize the file content. Non-text files are skipped. See Section A.4 for a detailed calculating procedure.

#Samples	Avg. #File	Avg. #Token
91	2.27	2.96×10^7

Table 2: Test case statistics by file and issue count.

	Single-File	Multi-File
Single-Issue	19	54
Multi-Issue	6	12

Table 3: Statistics of four issue tags in DCA-Bench. Each test case can contain more than one tag. See a full statistics in Section A.3.

data	document	infrastructure	ethical-legal
71	28	13	4

3.2 Test Case Construction

The test cases in DCA-Bench are collected or modified from the real issues reported by users or maintainers of eight dataset platforms, including Hugging Face [2], BIG-Bench [4], Kaggle, OpenML, TensorFlow Dataset, Open-Data-Registry, Five-Thirty-Eight, and Graph Learning Benchmark (GLI) [48]. The construction of the test cases of the DCA-Bench involved the following two stages.

3.2.1 Issue Collection and Preprocessing

The issue collection and preprocessing stage ensures the selection of relevant and manageable dataset issues. The steps in this stage are summarized as follows:

1. **Manual Selection of Dataset Platforms and Issues**: We first select dataset platforms where users and maintainers can interact. For example, there is a "Discussion" section for each Kaggle dataset to discuss dataset-related issues, which helps collect meaningful data quality issues. We then manually collect data quality issues from these discussions.
2. **Issue Filtering and Modification**: We prioritize the issues that have received feedback from dataset maintainers. Otherwise, we manually verify the problems mentioned in the issues to ensure their validity. We exclude issues that are not directly related to dataset files uploaded by contributors (e.g., issues about Tensorflow or Hugging Face data loading APIs). For issues involving multiple sub-issues or file sizes over 512MB, we either discard them or split them into manageable sub-cases with minimal adjustments.
3. **File Downloading**: After gathering candidate issues, we download the dataset files from the version before the problems are fixed. To simulate real-world scenarios, the dataset files included in each test case are not limited to those with known problems; we also include some additional files in the dataset not directly relevant to the problem.

3.2.2 Hints Generation

Asking Curators to discover hidden issues from raw dataset files can be very challenging. To make it more manageable and test the Curators' capability in finer granularity, we generate different levels of hints to help the Curators locate the hidden issues. We apply GPT-4 with carefully designed prompts[3] to generate three levels of hints for each test case, in addition to the no-hint setting:

[2] See https://huggingface.co/datasets

[3] See Appendix A.6 for the exact prompts.

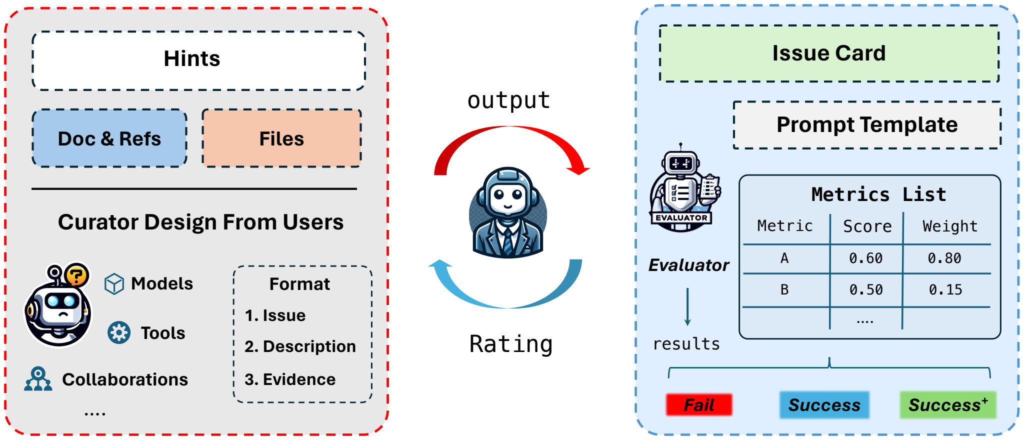

Figure 3: The Task I/O of DCA-Bench. For each test case, the input for the Curator includes dataset files and hints, with reference materials and platform documentation being optional. The Curator is then required to provide a description of the issue and corresponding contextual evidence. The label of the test case includes the issue title, content, the involved file names, and corresponding contextual evidence. Given the output from the Curator and the label, the Evaluator is then asked to rate the performance of the Curator.

- h_0: No hint provided. In this case, the Curator is required to detect the issue fully on its own.
- h_1: General description of the issue, without any specific details or hints on the location.
- h_2: Information about which files are involved in the issue, in addition to information from h_1.
- h_3: Partial contextual information about the issue, in addition to information from h_2.

After generating the hints, we manually double-check the hints and make necessary modifications to ensure that the generated hints follow the guidelines above.

3.3 Benchmark API

After gathering test cases and related information from dataset platforms, as well as generating multi-level hints, we proceed to develop the benchmark API, which encapsulates the inputs to the Curator and the Evaluator, adhering to the *Task I/O* paradigm as illustrated in Fig.3.

In this framework, the **Curator** receives following from benchmark API: (i) dataset files with hidden issues, (ii) hints on the issues, and (iii) optional dataset documentation and reference materials. It is then asked to identify, describe, and provide contextual evidence for these issues. The Curator is allowed to use any strategy and tools to process provided dataset files and find the issue, e.g., flattening the file contents and feeding them to the Curator, applying RAG technology, using a code interpreter to write and execute programs, or combining multiple strategies together.

The **Evaluator** then compares the outputs from the Curator with the label provided by the benchmark API to rate the performance of the Curator. The performance of the Curator is then classified into three levels: `fail`, `success`, and `success`$^+$, which we introduce in the following section.

3.4 Evaluation Pipeline

We now discuss the motivation and design of our evaluation pipeline. We evaluate the Curators in terms of the following two aspects: (i) whether the Curators accurately identify the issues, and (ii) whether the Curators provide necessary contextual evidence about the issue. We accordingly categorize the performance of the Curator into three levels:

- `fail`: The Curator fails to discover any issues, only identifies irrelevant issues, or acknowledges the issue but offers completely wrong contextual evidence.
- `success`: The Curator identifies the annotated issue and provides at least one correct piece of contextual evidence.

- `success`$^+$: The Curator correctly identifies all issues and provides all necessary contextual evidence.

Note that this evaluation process has several intrinsic challenges. Firstly, due to the nuanced nature of text generation, different Curator outputs can refer to the same issue, while similar Curator outputs may also point to different issues. Fixed keyword [49] or rule-based code tests [12] are often insufficient to distinguish the nuances. Additionally, Curators may be able to identify the issues using only a portion of the contextual evidence annotated in the test case, which means that evaluation protocols requiring the inclusion of all the annotated contextual evidence can be overly stringent. While human experts can easily capture such nuances and properly evaluate the Curator performance, human evaluation is too expensive for a benchmark in practice. Recognizing that evaluating the Curator's performance is easier than building the Curator agent, we propose an automatic evaluation pipeline where LLMs are leveraged to serve as the Evaluator.

We explored several prompting strategies using a few samples of human-annotated outcomes on a handful of issues and ended up with the following prompt design, which asks the LLMs to rate the Curator outputs in terms of a few criteria, and then aggregate these ratings to obtain an overall score for the Curator performance. The criteria we have in the Evaluator prompt are listed as follows:

- *Precise Contextual Evidence*: This criterion evaluates whether the Curator has detected and described the issue while providing accurate contextual evidence. The Curator should receive a low rating if it identifies the issue but fails to locate the correct context. A medium or high rating is justified only if the Curator identifies the issue and accurately pinpoints where it arises.
- *Detailed Issue Analysis*: This criterion examines whether the Curator's response extends beyond the mere repetition of provided hints, indicating a profound understanding of the issue.
- *Relevance of Reasoning*: This criterion ensures the Curator's logical reasoning is directly pertinent to the addressed problem rather than being generic or irrelevant.

For each test case, the Evaluator receives the Curator's output along with the hints and the ground truth annotations about the issues in the test case. Then, the Evaluator is asked to rate the Curator's performance for each criterion with a real number from 0 to 1. The final score is a weighted sum of the three ratings, with weights of 0.85, 0.15, and 0.05, reflecting their respective importance. The Evaluator then categorizes the Curator's performance according to defined thresholds on the final score. More details about the weights and thresholds are provided in Appendix A.7.

To enhance the robustness of the evaluation, a simplified voting strategy similar to Verga et al. [50] was implemented to mitigate the influence of randomness. The strategy comprises three rounds of voting. In the first round, we collect results from n Evaluators. If a consensus is reached, the decision is made with the consensus; however, if there is no consensus, we conduct another round of voting within m Evaluators, and apply a majority vote to make the decision. In the event of a tie, additional votes will be conducted one by one until a definitive majority vote is determined. In practice, m and n are both set to 2.

The testing results of the Evaluator on larger scale samples are displayed in Section 4.1, demonstrating its reliable performance. Notably, we collect annotated test data for the Evaluator *after* we have finalized the prompt design for our Evaluator. Therefore, we believe the proposed prompt design of our Evaluator can generalize across different Curators and issues in the DCA-Bench.

4 Experiments

In this section, we first verify the performance of the Evaluator, which we will show that it reliably aligns with human annotators. We then apply this Evaluator to test some baseline Curators on DCA-Bench and discuss the results.

4.1 Validation of the Evaluator

Experimental Setup To verify the effectivene of the proposed Evaluator, we compare it with human annotations. We randomly select 23 test cases for each of the four hint levels, which results in 92 (issue, hint) pairs. We then collect the outputs of a baseline Curator agent, which is based on the OpenAI Assistant API with `GPT-4-0125 preview` equipped with the Code Interpreter tool[4], on each of the 92 pairs. Then, we have the Evaluator and human annotators to rate the Curator outputs. For the Evaluator, we test four different models as the backend: `GPT-4-0125 preview`, `GPT-4o`, `GPT-3.5-turbo`, and `Llama3-70B-Instruct`. We use the `chatCompletion` API for OpenAI models and the Hugging Face interface for locally deploying Llama-3. For human annotators, we provide the codebook (see Appendix A.2) to follow. Finally, we compare the Evaluator's ratings with the human annotations (ground truth), focusing on the binary classification between "`fail`" and (`success`/`success`$^+$). For clarity, we refer to the latter as "`succ`" in the following context. The evaluation results using a three-class classification of "`fail`", `success`, and `success`$^+$ are provided in Appendix A.5.

Table 4: The performance of Evaluators with different backend models, using human annotations as ground truth. Here, we treat `succ` as the positive class when calculating Precision, Recall, and F1 Score. The "κ Value" refers to Cohen's κ between the Evaluator ratings and the human annotations.

	Success Rate / %				
Model Name	Accuracy	Precision	Recall	F1 Score	κ Value
`GPT-4-0125-preivew`	97.83	93.10	100.00	96.43	94.87
`GPT-4o`	92.39	81.25	96.30	88.14	82.59
`GPT3.5-turbo`	68.48	48.21	100.00	65.06	42.15
`Llama3-70B-Instruct`	65.22	45.61	96.30	61.90	36.69

Results As shown in Tab. 4, while the Evaluators with all backend models have high recall scores, which means they rarely misclassify `succ` as `fail`, it is quite often for weaker models (`GPT3.5-turbo` and `Llama3-70B-Instruct`) to misclassify `fail` as `succ`, as evidenced by the relatively low precision score.

Overall, the Evaluator with the backend `GPT-4-0125-preview` highly aligns with the human annotations, indicating that it could serve as a good proxy of human evaluations in our benchmark.

4.2 Benchmarking the Baseline Curators

Experimental Setup We apply our Evaluator with backend `GPT-4-0125-preview` to benchmark the performance of some baseline Curators. We first experiment with the baseline Curator based on the OpenAI Assistant API equipped with the Code Interpreter tool on the full set of test cases and hint levels. As pointed out by OpenAI API users[5], the OpenAI Assistant API's performance is often worse than that of the web-interface ChatGPT, so we also test Curators based on the web-interface ChatGPT. However, experimenting with the web-interface ChatGPT is not scalable, as we need to manually feed the prompts to the browser. Therefore, we selected 32 (issue, hint) pairs where the Assistant-API-based Curator fails, forming a hard set of DCA-Bench. Then, we carry out this small-scale experiment on ChatGPT-4, ChatGPT-4o, and ChatGPT-4 with reference materials. The model's knowledge is limited to May 2024, when we conduct the experiments. The performance of the Curator is classified as `fail` and `succ` by the Evaluator.

Results Fig. 4 shows the results of the Assistant-API-based Curator[6]. Without any hints (Hint Level 0), the Curator only successfully detects 10.99% of issues. With more informative hints, the performance of the Curator increases accordingly, achieving a 62.64% success rate when given the most informative hint (Hint Level 3). Despite this improvement, the success rate is still unsatisfactory given the amount of information provided, as we cannot expect to have such informative hints when developing the real autonomous dataset curation pipeline.

Tab. 5 shows the results of web-interface ChatGPT-based Curators on the hard set. As we can see, OpenAI's most powerful models, ChatGPT-4 and ChatGPT-4o, only succeeded in detecting approximately 19% of the issues. Intuitively, dataset curation knowledge should help the assistant better identify the issue. Therefore, we used OpenAI's MyGPT[7] to add reference materials in

[4]See https://platform.openai.com/docs/assistants/tools/code-interpreter.

[5]see Section A.8 for details

[6]See Appendix A.9 for a cost analysis of evaluating baseline Curator.

[7]https://openai.com/index/introducing-gpts/

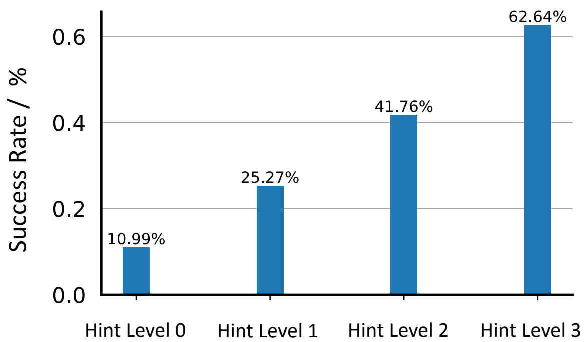

Figure 4: Results of the baseline Curator based on the OpenAI Assistant API.

Table 5: Success rates of different models tested on the hard set of DCA-Bench. Even strong language models like ChatGPT-4 and ChatGPT-4o only succeeded in detecting approximately 19% of the issues. Interestingly, providing ChatGPT-4 with reference materials led to a drop in its performance.

Model	**Succ /%**
`ChatGPT-4`	18.80
`ChatGPT-4o`	18.80
`ChatGPT-4 w/refs`	12.50

DCA-Bench as knowledge to ChatGPT and tested its performance (the row `ChatGPT-4 w/ refs`). Surprisingly, the performance of the model drops in comparison to the model without such extra knowledge. We suspect that this is because the reference materials introduce too generic information, taking up the context window of the model and reducing its attention to the context relevant to the specific issues.

Overall, the results of the baseline Curators suggest that the proposed DCA-Bench is a challenging benchmark for state-of-the-art LLMs.

5 Conclusion

To help the development of LLM agents capable of dataset curation, we present DCA-Bench, a collection of representative dataset issue cases from popular dataset platforms. Instead of fixing predefined issues, DCA-Bench aims to test the agent's capability to discover hidden data quality issues, a critical initial step in the dataset curation pipeline. To efficiently and effectively evaluate the performance of the Curator agents, we develop an LLM-based Evaluator with carefully designed prompts, which aligns well with human annotators. We conduct a benchmark study on the baseline Curator using DCA-Bench, and the results indicate that while LLMs have potential in real-world dataset curation, further exploration and innovation are needed to fully realize their capabilities.

Limitations Dataset curation is a complex and comprehensive problem, and the representative cases we collect might not fully cover the entire problem set. Additionally, due to the complexity of dataset files, we cannot guarantee that all the issues in the dataset files in DCA-Bench are labeled. Lastly, we haven't considered other modality information such as images or audios, which may be helpful to effectively curate multimedia datasets. Based on our work, future studies could explore developing more complex LLM agent systems, best practices for handling multi-modal information in dataset curation, or creating a more realistic simulation environment for testing LLM agents.

References

[1] Abhinav Jain, Hima Patel, Lokesh Nagalapatti, Nitin Gupta, Sameep Mehta, Shanmukha Guttula, Shashank Mujumdar, Shazia Afzal, Ruhi Sharma Mittal, and Vitobha Munigala. Overview and importance of data quality for machine learning tasks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 3561–3562, 2020.

[2] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020.

[3] Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger,

Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander M. Rush, and Thomas Wolf. Datasets: A community library for natural language processing, 2021.

[4] Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R. Brown, and et al. Adam Santoro. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models, 2023.

[5] Xinyu Yang, Weixin Liang, and James Zou. Navigating dataset documentations in ai: A large-scale analysis of dataset cards on hugging face, 2024.

[6] Jan-Christoph Klie, Richard Eckart de Castilho, and Iryna Gurevych. Analyzing dataset annotation quality management in the wild, 2024.

[7] Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé III au2, and Kate Crawford. Datasheets for datasets, 2021.

[8] Zige Wang, Wanjun Zhong, Yufei Wang, Qi Zhu, Fei Mi, Baojun Wang, Lifeng Shang, Xin Jiang, and Qun Liu. Data management for large language models: A survey, 2023.

[9] Ruyi Gan, Ziwei Wu, Renliang Sun, Junyu Lu, Xiaojun Wu, Dixiang Zhang, Kunhao Pan, Junqing He, Yuanhe Tian, Ping Yang, Qi Yang, Hao Wang, Jiaxing Zhang, and Yan Song. Ziya2: Data-centric learning is all llms need, 2024.

[10] Nitin Gupta, Hima Patel, Shazia Afzal, Naveen Panwar, Ruhi Sharma Mittal, Shanmukha Guttula, Abhinav Jain, Lokesh Nagalapatti, Sameep Mehta, Sandeep Hans, Pranay Lohia, Aniya Aggarwal, and Diptikalyan Saha. Data quality toolkit: Automatic assessment of data quality and remediation for machine learning datasets, 2021.

[11] Ze Mao, Yang Xu, and Erick Suarez. Dataset management platform for machine learning, 2023.

[12] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues?, 2024.

[13] Unit Mesh. auto-dev. `https://github.com/unit-mesh/auto-dev`, 2024.

[14] Wei Tao, Yucheng Zhou, Wenqiang Zhang, and Yu Cheng. Magis: Llm-based multi-agent framework for github issue resolution, 2024.

[15] Eileen S Jay. *The nature of problem finding in students' scientific inquiry*. Harvard University, 1996.

[16] Wikipedia contributors. Problem finding — Wikipedia, the free encyclopedia, 2024. URL `https://en.wikipedia.org/w/index.php?title=Problem_finding&oldid=1122334455`. [Online; accessed 18-April-2024].

[17] OpenAI. Gpt-4 technical report, 2024.

[18] Jan-Christoph Klie, Bonnie Webber, and Iryna Gurevych. Annotation error detection: Analyzing the past and present for a more coherent future. *Computational Linguistics*, 49(1):157–198, 2023.

[19] Leon Weber-Genzel, Robert Litschko, Ekaterina Artemova, and Barbara Plank. Donkii: Can annotation error detection methods find errors in instruction-tuning datasets? *arXiv preprint arXiv:2309.01669*, 2023.

[20] Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel R. Bowman, and Noah A. Smith. Annotation artifacts in natural language inference data, 2018.

[21] Youdi Gong, Guangzhen Liu, Yunzhi Xue, Rui Li, and Lingzhong Meng. A survey on dataset quality in machine learning. *Information and Software Technology*, 162:107268, 2023. ISSN 0950-5849. doi: https://doi.org/10.1016/j.infsof.2023.107268. URL `https://www.sciencedirect.com/science/article/pii/S0950584923001222`.

[22] Tong Zhou, Yubo Chen, Pengfei Cao, Kang Liu, Jun Zhao, and Shengping Liu. Oasis: Data curation and assessment system for pretraining of large language models, 2023.

[23] Maha Jarallah Althobaiti. Bert-based approach to arabic hate speech and offensive language detection in twitter: exploiting emojis and sentiment analysis. *International Journal of Advanced Computer Science and Applications*, 13(5), 2022.

[24] Hannah Rose Kirk, Abeba Birhane, Bertie Vidgen, and Leon Derczynski. Handling and presenting harmful text in nlp research, 2023.

[25] Wei-Chen Wang and Jonas Mueller. Detecting label errors in token classification data. *arXiv preprint arXiv:2210.03920*, 2022.

[26] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M. Zhang. Large language models for software engineering: Survey and open problems, 2023.

[27] Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. Codet: Code generation with generated tests, 2022.

[28] Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. Wizardcoder: Empowering code large language models with evol-instruct, 2023.

[29] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code, 2024.

[30] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021.

[31] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021.

[32] Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Scott Wen tau Yih, Daniel Fried, Sida Wang, and Tao Yu. Ds-1000: A natural and reliable benchmark for data science code generation, 2022.

[33] Yakun Zhang, Wenjie Zhang, Dezhi Ran, Qihao Zhu, Chengfeng Dou, Dan Hao, Tao Xie, and Lu Zhang. Learning-based widget matching for migrating gui test cases. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, ICSE '24. ACM, February 2024. doi: 10.1145/3597503.3623322. URL `http://dx.doi.org/10.1145/3597503.3623322`.

[34] Yangruibo Ding, Zijian Wang, Wasi Uddin Ahmad, Hantian Ding, Ming Tan, Nihal Jain, Murali Krishna Ramanathan, Ramesh Nallapati, Parminder Bhatia, Dan Roth, and Bing Xiang. Crosscodeeval: A diverse and multilingual benchmark for cross-file code completion, 2023.

[35] Tianyang Liu, Canwen Xu, and Julian McAuley. Repobench: Benchmarking repository-level code auto-completion systems, 2023.

[36] Yonghao Wu, Zheng Li, Jie M. Zhang, Mike Papadakis, Mark Harman, and Yong Liu. Large language models in fault localisation, 2023.

[37] Sidong Feng and Chunyang Chen. Prompting is all you need: Automated android bug replay with large language models, 2023.

[38] Spandan Garg, Roshanak Zilouchian Moghaddam, and Neel Sundaresan. Rapgen: An approach for fixing code inefficiencies in zero-shot, 2024.

[39] Zimin Chen, Sen Fang, and Martin Monperrus. Supersonic: Learning to generate source code optimizations in c/c++, 2023.

[40] Russell A Poldrack, Thomas Lu, and Gašper Beguš. Ai-assisted coding: Experiments with gpt-4, 2023.

[41] Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-eval: Nlg evaluation using gpt-4 with better human alignment, 2023.

[42] Cheng-Han Chiang and Hung-yi Lee. Can large language models be an alternative to human evaluations? In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15607–15631, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.870. URL `https://aclanthology.org/2023.acl-long.870`.

[43] Hangfeng He, Hongming Zhang, and Dan Roth. Socreval: Large language models with the socratic method for reference-free reasoning evaluation, 2024.

[44] Shibo Hao, Yi Gu, Haotian Luo, Tianyang Liu, Xiyan Shao, Xinyuan Wang, Shuhua Xie, Haodi Ma, Adithya Samavedhi, Qiyue Gao, Zhen Wang, and Zhiting Hu. Llm reasoners: New evaluation, library, and analysis of step-by-step reasoning with large language models, 2024.

[45] Yixiong Chen, Li Liu, and Chris Ding. X-iqe: explainable image quality evaluation for text-to-image generation with visual large language models, 2023.

[46] Zhiyuan You, Zheyuan Li, Jinjin Gu, Zhenfei Yin, Tianfan Xue, and Chao Dong. Depicting beyond scores: Advancing image quality assessment through multi-modal language models, 2024.

[47] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021.

[48] Jiaqi Ma, Xingjian Zhang, Hezheng Fan, Jin Huang, Tianyue Li, Ting Wei Li, Yiwen Tu, Chenshu Zhu, and Qiaozhu Mei. Graph learning indexer: A contributor-friendly and metadata-rich platform for graph learning benchmarks, 2022.

[49] Qianyu He, Jie Zeng, Wenhao Huang, Lina Chen, Jin Xiao, Qianxi He, Xunzhe Zhou, Lida Chen, Xintao Wang, Yuncheng Huang, Haoning Ye, Zihan Li, Shisong Chen, Yikai Zhang, Zhouhong Gu, Jiaqing Liang, and Yanghua Xiao. Can large language models understand real-world complex instructions?, 2024.

[50] Pat Verga, Sebastian Hofstatter, Sophia Althammer, Yixuan Su, Aleksandra Piktus, Arkady Arkhangorodsky, Minjie Xu, Naomi White, and Patrick Lewis. Replacing judges with juries: Evaluating llm generations with a panel of diverse models, 2024.

[51] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

[52] Aobo Kong, Shiwan Zhao, Hao Chen, Qicheng Li, Yong Qin, Ruiqi Sun, Xin Zhou, Enzhi Wang, and Xiaohang Dong. Better zero-shot reasoning with role-play prompting, 2024.

[53] OpenAI. Six strategies for getting better results. `https://platform.openai.com/docs/guides/prompt-engineering/six-strategies-for-getting-better-results`, 2024. Accessed: 2024-05-28.

A Appendix

A.1 Real World Dataset Curation Examples

The real-world dataset curation process can be complex. During the dataset contribution procedure, authors and administrators must communicate iteratively to address issues in the contributed datasets. These issues can be numerous and challenging to address comprehensively at once, even for proficient individuals. Additionally, new issues often arise while fixing existing ones, leading to a complex contribution procedure. Below, we provide a few URLs to real-world cases on the TensorFlow dataset and BIG-Bench to illustrate these challenges:

- https://github.com/tensorflow/datasets/pull/1360
- https://github.com/tensorflow/datasets/pull/1549
- https://github.com/google/BIG-bench/pull/870

Besides, dataset issues are often reported by dataset users, even though uploaded datasets have passed the initial checks of the administrators. Here are some representative examples classified by tags:

Insufficient documentation

- https://github.com/Graph-Learning-Benchmarks/gli/issues/259
- https://www.kaggle.com/datasets/pkdarabi/brain-tumor-image-dataset-semantic-segmentation/discussion/479324

Inaccurate annotations

- https://github.com/google/BIG-bench/issues/938
- https://github.com/google/BIG-bench/issues/872
- https://github.com/tensorflow/datasets/issues/1207

Ethical concerns

- https://github.com/google/BIG-bench/pull/685
- https://www.kaggle.com/datasets/vikrishnan/boston-house-prices/discussion/429030
- https://www.kaggle.com/datasets/pavansubhasht/ibm-hr-analytics-attrition-dataset/discussion/157179

Cross-file discrepancies

- https://www.kaggle.com/datasets/antonkozyriev/game-recommendations-on-steam/discussion/4200733"
- https://www.kaggle.com/datasets/nelgiriyewithana/global-youtube-statistics-2023/discussion/438729
- https://www.kaggle.com/datasets/sudarshan24byte/online-food-dataset/discussion/491973

A.2 Human Annotation Codebook

The human annotators are required to follow the following codebook when rating the responses from the Curators:

You are tasked with evaluating the work of a Dataset Curator, who is responsible for identifying issues in dataset files and pinpointing the contextual evidence within those files. You will be provided with a ground truth that specifies the issues present in the files and their exact locations.

Based on this information, please categorize the Curator's findings into the three following categories:

- `fail`: the Curator either denies the existence of the issue stated on the label, identifies irrelevant issues, or acknowledges the issue but offers completely inaccurate supporting evidence.

- `success`: The Curator identifies the annotated issue and provides at least one correct piece of contextual evidence.
- `success`$^{+}$: the Curator correctly identifies all issues and provides all necessary contextual evidence.

Additional Rules: 1. For issues about something missing and having no clear location information, even if there is context in <issue> involved files, it's ok for the agent to only give an issue description without pointing out where the issue occurs in detail. 2. You should focus on whether the agent has spotted the issue in ground truth rather than caring about whether the agent includes unrelated examples not present in the context. The inclusion of other irrelevant issues won't influence its performance in evaluation.

A.3 Issue Type and Tag

We classify the issues into single-issue / multi-files and single-file/multi-files.

- **single-issue**: This sample has only one known issue.
- **multi-issue**: this sample has multiple issues. If many issues of the same type exist in a sample, we also classify them into multi-issue.
- **single-file**: the environment of this sample has only one file
- **multi-file**: the environment of this sample has multiple files

To help benchmark users better understand the issue, we assigned each of them several tags. Some tags have a structure of affiliations.

typo: Issues caused by typographical errors.

wrong-format: Problems related to incorrect formatting.

inappropriate-file: Missing, empty or redundant files.

ethical/legal-risk: Ethical or legal risks associated with the dataset, such as racial bias, missing license.

cross-file-discrepancy: Discrepancies between files (e.g., meta-information in the documentation does not match the actual data).

internal-discrepancy: Discrepancies within a single file.

data-problem: Issues related to the data files.

|- **wrong-value**: Errors in the data values.
|- **missing-value**: there are missing columns or values.
|- **data-leakage**: Risks of data leakage.
|- **apparent-corruption**: Errors in the data files that can be easily detected using the information within the file. (e.g. duplicated data, apparently wrong target format compared to other targets in the same file, CSV file with wrong format)
|- **hidden-corruption**: Errors in the data files that require external knowledge and logical reasoning to resolve.

document-problem: Issues related to the documentation files(e.g. README, DataCard, meta-data and other descriptive text) of the dataset.

|- **wrong-info**: Wrong information in the documentation files (e.g., wrong meta-information, typos, invalid URLs, and email addresses).
|- **insufficient-info**: Missing or unclear information in the documentation files (e.g., meanings of columns, labels, units of data values).

infrastructure-problem: Issues with fetching, loading, processing, or displaying data.

|- **data-access**: Problems accessing the data.
|- **script-code**: Issues with scripts.

Tab. 6 displays the number of each tag in DCA-Bench in detail.

Table 6: The number of tags in DCA-Bench. Note that the sum of the sub-category might not equal the parent category. For instance, if there's an ethical or legal risk identified in a dataset document, it should be tagged as ["document-problem", "ethical/legal-risk"], without including any sub-categories of "document-problem" that are not relevant.

Category	Number	Sub-category	Number
typo	13	—	—
wrong-format	6	—	—
inappropriate-file	4	—	—
ethical/legal-risk	4	—	—
internal-discrepancy	11	—	—
cross-file-discrepancy	23	—	—
data-problem	71	wrong-value	26
		missing-value	5
		data-leakage	2
		apparent-corruption	10
		hidden-corruption	25
document-problem	28	wrong-info	17
		insufficient-info	8
infrastructure-problem	16	data-access	2
		script-code	14

A.4 Calculation of File Number and Context Length

We calculate the file context length using the `tiktoken` package[8] to tokenize the file content, using:

```
import tiktoken

def calculate_token_len(sentence, model="gpt-4-0125-preview"):
    encoding = tiktoken.encoding_for_model(model)
    tokens = encoding.encode(sentence)
    token_len = len(tokens)
    return token_len
```

During calculation, non-text files are skipped, which include .jpg, .jpeg, .png, .gif, .bmp, .mp3, .wav, .ogg, .flac, .mp4, .avi, .mov, .mkv.

For .zip files, we will unzip them to calculate the context length of text files. However, when calculating the file number, we take the whole .zip number as one single file.

A.5 Strict Evaluation

As defined in Section 3.3, the performance of the Curator is categorized into `fail`, `success`, and `strict-success`. However, accurately distinguishing between `success` and `strict-success` in the original triple classification is challenging for Evaluators.

Tab. 7 presents the complete results of the Evaluators' performance with strict success alignment, compared to human annotations (ground truth). There is a significant drop in the Evaluators' performance when required to provide a triple classification.

A.6 Prompt of Hints Writer

The Hints Writer is prompted with the following instructions to generate three level hints for the issue, with `<ISSUE>` being the placeholder.

[8]https://github.com/openai/tiktoken

Table 7: The performance of Evaluators with different back-end models, using human annotations as ground truth on 92 (issue, hint) pairs. Compared with binary classification (`fail` / (`success`+`success`$^+$), the triple classification proves to be more difficult, making the evaluation results from the Evaluator less reliable.

	Binary / %					Triple / %	
Model Name	Accuracy	Precision	Recall	F1 Score	κ value	Accuracy	κ value
`GPT-4-0125-preivew`	97.83	93.10	100.00	96.43	94.87	89.13	76.52
`GPT-4o`	92.39	81.25	96.30	88.14	82.59	83.70	66.47
`GPT3.5-turbo`	68.48	48.21	100.00	65.06	42.15	63.04	38.82
`LLama3-70B-Instruct`	65.22	45.61	96.30	61.90	36.69	59.78	35.26

```
You are an issue-hint writer. You need to formulate some hints for detecting certain
     issues, based on the description in
<issue> below.

<rules>
- The hints should have three different levels, from very general to very specific.
- For ‘‘general hint’’, you should only tell the general issue, without any specific
     information.
- For ‘‘medium hint’’, you could tell which files are involved in this issue +
    general hint. Make sure it contains the information from the general hint.
- For ‘‘specific hint’’, you should tell which files are involved in this issue
    clearly, as well as a part of context information. Make sure it contains the
    information from the general and medium hints.
- You should never tell the agent the full context of the issue, only give it hints,
     so we can test to what degree the agent detects the issue and tell us evidence
     by themselves.
- Your word used in hints should be precise, pay attention to the word choice, and
    try to avoid misleading or ambiguous words. You should make sure you have a
    clear understanding of the issue before writing the hints.
- The hints should be clear and concise, try to cover all the issues mentioned and
    avoid unnecessary information.
</rules>

<example>
{
    <issue>
    {
        title: What is the provenance of these benchmark datasets?}
        content:
            {
                Firstly, check the English proverbs dataset: https://github.com/
                    google/BIG-bench/tree/main/bigbench/benchmark_tasks/
                    english_proverbs
                ‘‘‘
                    task can help to draw conclusions about human-like understanding
                        and
                    ## Data source
                    ## References
                ‘‘‘
                There is nothing in the data source section.
            }
        involved: {
            name: README.md,
            context: {
                ‘‘‘
                    task can help to draw conclusions about human-like understanding
                        and Data Source References
                ‘‘‘
            }
```

```
        }
    }
    </issue>

    output:
    {
        general: "section with empty content"
        medium: "a section with empty content in markdown file"
        specific: "a sub-heading section in README.md with empty content"
    }

</example>

------

Now let's begin

<issue>
<ISSUE>
</issue>

write your hints here in JSON format:
```

When designing the prompt, we take into consideration the risk that hints could provide too much information that can already serve as a valid answer. In the "<rule>" section, we clearly require the Hint Writer not to reveal the full context of the issue to the Curator, in case it takes it as a shortcut and skips reading the real files. We also manually check the generated hints to ensure the generated hints comply this rule.

A.7 Design Details of the Evaluator

In this part, we elaborate on the prompt design of our Evaluator. While it's natural to ask LLMs to directly classify the Curators' performance into `fail`/`success`/success^{+}, this strategy is poorly aligned with the annotation of human experts as we tested. In addition to common prompt methods (e.g., COT [51], role-playing [52], delimiter [53]), we believe more prompt engineering is needed to improve the performance of the Evaluator, so we design a metrics-rating strategy to enhance the accuracy of the Evaluator.

The motivation behind this design is to make iterative refinement of prompts more effective. Developing a good prompt requires testing it on a subset of human-labeled issues with corresponding outcomes from Curators and then making adjustments based on the outcomes. Unlike the model training process, which benefits from clear gradients and systematic optimization, tuning text prompts is often an empirical process necessitating numerous trials without the guarantee of improvement in its performance.

On the contrary, employing numerical weights and thresholds enables us to tune the prompt more fine-grained, only changing the weights or threshold based on the results, which share similarities with linear regression, resulting in a more efficient prompt tuning process and better performance. In this way, we finalize

To be more detailed, the Evaluator is prompted with the following instructions to evaluate the performance of the agent, with `<ISSUE>`, `<HINT>`, and `<ANSWER>` being the place-holder.

```
You are required to act as an answer evaluator. Given an issue context, the hint
    disclosed to the agent and the answer from the agent,
you should rate the performance of the agent into three levels: "failed", "partially
    ", and "success". The rating rules are as follows:

<rules>
1. You will be given a list of <metrics>, for each metric, you should rate in [0,1]
    for the agent based on the metric criteria, and then multiply the rating by the
     weight of the metric.
```

```
2. If the sum of the ratings is less than 0.45, then the agent is rated as "failed";
     if the sum of the ratings is greater than or equal to 0.45 and less than 0.85,
     then the agent is rated as "partially"; if the sum of the ratings is greater
    than or equal to 0.85, then the agent is rated as "success".
3. **<text>** means the text is important and should be paid attention to.
4. ****<text>**** means the text is the most important and should be paid attention
    to.
</rules>

The <metrics> are as follows:
{
    "m1": {
        "criteria": "Precise Contextual Evidence:
            1. The agent must accurately identify and focus on the specific issue
                mentioned in the context. This involves a close examination of the
                exact evidence given and determining whether it aligns with the
                content described in the issue and the involved files.
            2. Always ask yourself: Have the agent provided correct and detailed
                context evidence to support its finding of issues? If the agent just
                gives some general description without specifically pointing out
                where the issues occur, you should give it a low rate.
            3. Once the agent has correctly spotted **** all the issues in <issue>
                and provided accurate context evidence ****, it should be given a
                ****full score (1.0) even if it includes other unrelated issues/
                examples ****"
            4. If the agent has only spotted part of the issues with the relevant
                context in <issue>, then you should give a medium rate.
            5. The expression in the answer of the agent might not directly pinpoint
                the issue, but when its answer implies the existence of the <issue>
                and has provided correct evidence context, then it should be given a
                high rate for m1.
            6. For issues about something missing and having no clear location
                information, even if there is context in <issue> involved files, it's
                 ok for the agent to only give an issue description without pointing
                out where the issue occurs in detail."
         "weight": 0.8,
        "range": [0, 1]
    },
    "m2": {
        "criteria": "Detailed Issue Analysis:
            1. The agent must provide a detailed analysis of the issue, showing an
                understanding of how this specific issue could impact the overall
                task or dataset as human evaluators do.
            2. This metric stresses the importance of not just identifying that there
                 is an issue but also understanding and explaining its implications
                in detail.",
        "weight": 0.15,
        "range": [0, 1]
    },
    "m3": {
        "criteria": "Relevance of Reasoning:
            1. The agent's reasoning should directly relate to the specific issue
                mentioned, highlighting this inconsistency's potential consequences
                or impacts.
            2. This metric ensures that the agent's logical reasoning directly
                applies to the problem at hand."
        "weight": 0.05,
        "range": [0, 1]
    }
}
</metrics>

--------------------------------

Now let's begin:
```

```
<issue>
<ISSUE>
</issue>

<hint>
<HINT>
</hint>

-------------------- Below is the answer from the agent. Ensure you don't take the
    information above as the agent's answer!

<answer>
<ANSWER>
</answer>

--------------------

response below,
1. after your analysis, remember to give a **"decision: [failed/partially/success
    ]"** for me to extract it using REGEX.
2. Don't use Code Interpreter!; Use your ability to analyze the text. ** Pay
    attention to your calculations and make sure they are correct. **
3. There could be multiple issues described in <issue> part. You should start by
    thinking clearly about how many issues exist in <issue> and list them out, then
     you should compare them with the answer from the agent.
4. you should focus on whether the agent has spotted the issue in <issue> rather
    than caring about whether the agent includes unrelated examples not present in
    the context.
```

Please note the categories of the Curator's performance are different from papers. We use `partially` for `success` in the paper. This modification is made because we want to emphasize we focus more on separating `fail` from the other two categories, which is a binary classification problem. Using `partially` might cause confusion.

A.8 Discussion on the Difference between OpenAI Assistant API and ChatGPT

The performance discrepancy between OpenAI's Assistant API and ChatGPT has been widely discussed within the OpenAI user community. Users often report that the Assistant API's performance does not match the capabilities of ChatGPT, especially in tasks involving complex file processing and analysis.

The Assistant API provides several methods for handling files: 1. Code Interpreter for file uploads. 2. Building VectorStores via File Search. 3. Attaching files to messages using their file IDs.

However, these methods have significant issues: - The quality of the VectorStore built using File Search is inferior to ChatGPT's built-in file search capabilities. - Files are referenced by their file IDs rather than their names, leading to confusion and errors during processing.

The following code snippet illustrates the process of uploading and referencing files in the Assistant API by attaching them to the messages:

```python
msg_file_ids = []
for file_path in file_paths:
    message_file = self.client.files.create(
        file=open(file_path, "rb"), purpose="assistants"
    )
    msg_file_ids.append(message_file.id)

attachments = [{"id": file_id, "tools": [{"type": "file_search"}]} for file_id in
    msg_file_ids]

thread = self.client.beta.threads.create(
```

```
    messages=[
        {
            "role": "user",
            "content": input,
            "attachments": attachments
        }
    ]
)
```

Despite correctly assigning file names during the upload process, the Assistant API seems to ignore these names during subsequent file processing, relying solely on file IDs. This results in frequent mix-ups and incorrect file references.

Several forum posts and articles highlight deficiencies relevant to this topic: - Different responses assistant playground vs API - using the same assistant ID - Huge difference between ChatGPT assistant and API assistant - Assistant API not consistent. This implies the limitations of OpenAI Assistant API.

In conclusion, while the Assistant API offers various tools for convenient file handling, its current implementation presents significant challenges that hinder its effectiveness compared to ChatGPT, which motivates us to select a subset of DCA-Bench containing hard cases to test on ChatGPT.

A.9 Cost Analysis

In this section, we provide an estimated cost analysis for applying our Evaluator backend by `GPT-4-0125-preview` to rate the performance of the Baseline Curator (OpenAI Assistant API) on DCA-Bench (394 inputs in total). For tokenization of contexts, we utilized `tiktoken`[9].

Based on the pricing information available as of May 2024 from https://openai.com/api/pricing/, the costs are as follows:

- Input cost: $6.95
- Output cost: $12.82
- Total cost: $19.77

The costs are rounded to two decimal places. Note the cost here only involves the Evaluator; we dismiss the cost to run the baseline Curator on DCA-Bench to get the outputs, which depends on the design of the Curator and thus is not our focus.

[9] https://github.com/openai/tiktoken

A.10 Issue Examples

Table 8: An issue example reported on Kaggle which involves racial bias

Title `Boston House Prices B feature is RACIST`

Meta-Info

- ID: `7e8f31cb-8c2a-4676-b3d4-941a64184a26`
- Platform: `Kaggle`
- Issue Type: single-issue & multi-file
- Issue Tags: ethical-legal-risk document-problem
- Source: `https://www.kaggle.com/datasets/vikrishnan/boston-house-prices/discussion/429030`

Content

`B: 1000(Bk-0.63)2 where Bk is the proportion of blacks by town No other race is featured in this dataset. Red-lining anyone?`

Involved Files

- name: `datacard.md`
- context: `PTRATIO:pupil-teacher ratio by town 12. B: 1000(Bk-0.63)2 where Bk is the proportion of blacks by town 13. LSTAT:% lower status of the population`

Hints

h_1 dataset contains potentially biased feature

h_2 bias in a feature documented in a markdown file

h_3 a feature in datacard.md is described using a formula that appears to single out one race

Table 9: An issue example reported on BIG-Bench that involves a discrepancy between dataset files.

Title `Miss aligned static information`

Meta-Info

- ID: `80d6db6a-6cbf-4261-8d13-3244e7fb54fd`
- Platform: `BIG-Bench`
- Issue Type: single-issue & multi-file
- Issue Tags: cross-file-discrepancy, document-problem/wrong-info
- Source: `https://github.com/google/BIG-bench/pull/498`

Content

`The stastic info in README.md is not aligned with the actual data file. There are 190 stories rather than 194 stories; 99 Ÿesr̈ather than 100 Ÿes;̈ 91 N̈or̈ather than 94 N̈o.̈`

Involved Files

1. name: `task.json`
   - context: `the number of datapoints in data files.`
2. name: `README.md`
   - context: `We collected 194 stories from 30 papers published in the span of 1989 to 2021. Each story has a causal judgment question associated with it with a "Yes" or "No" answer. We carefully balanced the dataset - there are 100 "Yes" answers (52%) and 94 "No" answers (48%). Each paper that we collected from has conducted rigorous human experiments. We follow a simple binarization strategy to reflect the majority of human agreement and use it as the ground truth to evaluate the AI model.`

Hints

h_1 Mismatched quantitative information

h_2 Mismatched quantitative information in README.md and data file

h_3 Quantitative discrepancies between README.md and task.json, specifically in the counts of stories, 'Yes' answers, and 'No' answers

Table 10: An issue example which has a wrong target label that needs precise factual knowledge to discern

Title `Error in 118th Congress data`

Meta-Info

- ID: `51e12546-8bf3-473c-9ed6-f85d63c357ce`
- Platform: `FiveThirtyEight`
- Issue Type: single-issue & multi-file
- Issue Tags: data-problem/hidden-corruption, data-problem/wrong-value
- Source: `https://github.com/fivethirtyeight/data/issues/336`

Content

`The "congress-demographics" data includes Benjamin Eric Sasse as being a member of the 118th Congress but he resigned after the 117th.`

Involved Files

- name: `data_aging_congress.csv`
- context: `The "congress-demographics" data includes Benjamin Eric Sasse as being a member of the 118th Congress but he resigned after the 117th.`

Hints

h_1 inaccurate data entry

h_2 an inaccurate data entry in a CSV file

h_3 an entry in 'data_aging_congress.csv' inaccurately includes a member as part of the 118th Congress

Table 11: An issue example which has a wrong target label that requires translation ability to discern

Title `Mistranslation in conlang_translation task?`

Meta-Info

- ID: `bb29a2c3-872b-41cc-ac55-b26f22043da6`
- Platform: `BIG-Bench`
- Issue Type: single-issue & multi-file
- Issue Tags: data-problem/wrong-value, data-problem/hidden-corruption
- Source: `https://github.com/google/BIG-bench/issues/553`

Content

```
On the English to Gornam translation, one of the translations may be incorrect.
Specifically, the example given is English:  They want to eat my pizzas.  Gornam:
Sa wott min Pizzas atten.  However, from looking at the other examples, it seems
like when the subject is plural (like they), the suffix en is attached to the word,
so the correct Gornam translation should be Sa wotten min Pizzas atten.  Is this
true, or is there some flaw in this logic that we are missing?
```

Involved Files

- name: `task.json`
- context:

```
     "examples": [
        {"input": "I want to buy the orange.", "target": "Ek wott dei Orange leuren."},
        {"input": "He is wearing my pants.", "target": "Ha trugt min Rose."},
   -    {"input": "They want to eat my pizzas.", "target": "Sa wott min Pizzas atten."},
   +    {"input": "They want to eat my pizzas.", "target": "Sa wotten min Pizzas atten."},
        {"input": "I can eat.", "target": "Ek conn atten."},
        {"input": "They eat their shirts.", "target": "Sa atten hir Pemts."},
        {"input": "He tries on my coat.", "target": "Ha roturt min Enzu en."},
```

Hints

h_1 incorrect translation example

h_2 incorrect translation example in a JSON file

h_3 a potential translation inconsistency in 'task.json' involving plural subject examples