Minimization Of Large JSON Input For Efficient Debugging

Koteswara Rao Bade
Department of Computer Science
Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, NY 14623
kb5608@cs.rit.edu

***Abstract*—In this project, we present a novel approach to simplify the debugging process for developers working with large JSON lines data. Our solution involves the creation of a program that iteratively reduces the size of the JSON lines file by removing the JSON objects which are not responsible for the error, providing developers with a more manageable subset of the data. By progressively minimizing the input, we aim to improve the efficiency and effectiveness of debugging procedures significantly, ultimately streamlining the development workflow.**

I. Introduction

Debugging errors in large JSON lines data poses significant challenges for developers, often leading to lengthy and error-prone manual debugging processes. In response to this problem, our project focuses on developing a program that systematically minimizes input data, making it easier for developers to work with smaller subsets during the debugging phase. By dividing the input into halves, running the program on each subset, and evaluating the occurrence of errors, our approach progressively reduces the data to its smallest possible subset while retaining its integrity. This reduction process saves valuable developer time, facilitates quicker issue identification, and enhances the overall debugging experience. Through this report, we present our solution's architecture, implementation, and results, demonstrating its effectiveness in simplifying debugging workflows for developers dealing with large JSON lines data.

II. Background

JSON (JavaScript Object Notation) has become a widely adopted data interchange format in modern software development due to its simplicity, ease of use, and human-readable structure. It allows for structured data representation in key-value pairs, making it ideal for data exchange between applications and services. As data processing requirements continue to grow, so do the volume and complexity of JSON data, leading to the emergence of challenges in handling large datasets efficiently.

One of the specific challenges developers face is debugging errors in extensive JSON data, especially when the data is stored in JSON Lines (JSONL) format. JSON Lines extends the JSON format, where each line in a file represents a standalone JSON object. JSONL is advantageous for processing large datasets and streaming scenarios, allowing individual JSON objects to be read and processed independently.

Manual debugging of extensive JSON Lines data is time-consuming and prone to errors, especially when identifying the root cause of an issue within the data. To address this challenge and streamline the debugging process, we proposed a novel approach involving the development of a program that systematically reduces the size of the JSONL file.

The motivation behind this project lies in providing developers with a more manageable subset of the JSON Lines data, enabling them to focus on smaller portions of the input during the debugging phase. By iteratively dividing the input data, running the program on each subset, and evaluating errors, the reduction process aims to identify and present the smallest possible subset of the data that still exhibits the same issues. This reduced input allows developers to concentrate their efforts on a smaller, more focused dataset, significantly improving the efficiency and effectiveness of the debugging workflow.

Through this project, we aim to develop an automated solution that simplifies the debugging process for developers working with extensive JSON Lines data. The subsequent sections of this report will elaborate on the architecture and implementation, showcasing its effectiveness in reducing the complexity of debugging tasks and facilitating more streamlined development workflows.

III. Architecture And Implementation

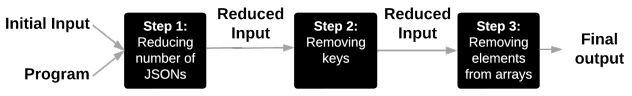

Fig. 1. Program Architecture

Initial Input:

In the context of our discussion, we will be focusing on understanding how the reduction program operates, and to illustrate this process, we will be using the specific input data depicted in figure 2. This example serves as a practical demonstration to help you grasp the functioning of

{"name": "Peach", "state": "NY", "prices": { "region_wise" : [10, 12, 13], "store_wise" : [8]}}
{"name": "Apple", "state": "CA", "prices": { "region_wise" : [13, 11, 12], "store_wise" : [7]}}
{"name": "Pears", "state": "NY", "prices": { "region_wise" : [10,"14",13], "store_wise" : [8]}}

Fig. 2. Initial input

the reduction program more effectively. By analyzing this particular input and observing the outcomes of the reduction process, we aim to provide a clear and tangible explanation of the program's behavior and its impact on the data.

Example Program:

```
def for_ss():
    total_prices = {}
    for line in sys.stdin:
        obj = json.loads(line)
        prices = obj['prices']['region_wise']
        if obj['state'] not in total_prices:
            total_prices[obj['state']] = prices[1]
        else:
            total_prices[obj['state']] += prices[1]
```

Fig. 3. Program example

The program shown in figure 3 will help us understand how things work. We'll use this program to explain the process that uses the data from figure 2. By looking at how the program interacts with the input from figure 2, we can see how the program carries out its tasks. This will give us a clear picture of how everything fits together and how the program handles the given data.

The upcoming sections will demonstrate the step-by-step functioning of the program, as illustrated in figure 1. We will utilize the example input from figure 2 and refer to the sample program in figure 3 to showcase each stage of the program's operation.

A. Reducing the number of JSONs in the input

In this step, the primary objective is to reduce the number of JSON objects in the input JSON lines file. The process is carried out iteratively, starting with the initial input, divided into two halves. The program runs the given program on each half separately and checks for any errors that might arise during the execution. Should an error be encountered in one of the halves, it signifies that the error likely originates from the corresponding section of the input data. Consequently, the program proceeds with the half that produced the error as the new input for the subsequent iteration, aiming to narrow down the problematic area further.

On the other hand, if no errors are detected in either of the halves, the program concludes that the entire input data does not contain the error. In this case, the program reshuffles the input data, seeking to expose different combinations and possible sources of the error. Throughout this iterative process, the number of JSON objects in the input is progressively reduced, and the program dynamically adapts the input data to identify the smallest subset that exhibits the error. This reduction process effectively narrows down the scope of the debugging task, providing developers with a more focused and manageable dataset to analyze.

The figure 4 illustrates the resulting reduced input obtained from this step, which subsequently becomes the input for the subsequent stages of the program.

{"name": "Peach", "state": "NY", "prices": { "region_wise" : [10, 12, 13], "store_wise" : [8]}}
{"name": "Pears", "state": "NY", "prices": { "region_wise" : [10,"14", 13], "store_wise" : [8]}}

Fig. 4. Reduced input after step 1

B. Removing unwanted keys from the input

In this step of the project, the focus is on removing unwanted keys from the input data in an iterative manner. The program successively eliminates keys from the input and tests the modified data against the target program. Should an error arise during this process, it signifies that the removed key is not responsible for the error, leading the program to deduce that the key can be safely eliminated from all JSON objects.

As an illustrative example, let us consider the key "name" unused within the program. Given this insight, the program removes the "name" key from all JSON objects in the input data. Post the key removal operation; the resulting input data will be updated to exclude the "name" key throughout the entire dataset. In addition to the "name" key, the "storewise" key is also not utilized in the program. Consequently, the "storewise" key will also be eliminated. By systematically removing irrelevant keys from the input data, this step effectively streamlines the dataset and narrows down the potential sources of errors. This reduction process plays a pivotal role in enhancing the debugging process by providing developers with a more concise and focused dataset, facilitating the identification of genuine issues, and eliminating unnecessary distractions.

The figure 5 shows the resulting reduced input obtained from this step, serves as the foundation for further refinement and reduction in subsequent phases of the program.

{"state": "NY", "prices": {"region_wise": [10, 12, 13]}}
{"state": "NY", "prices": {"region_wise": [10,"14",13]}}

Fig. 5. Reduced input after step 2

C. Removing unwanted elements from the array type keys

In this project's final step, the program aims to eliminate unwanted elements from keys of the type array in the input data. The program reduces the elements by half each iteration by employing an iterative approach similar to step 1. By systematically evaluating the execution results, the program identifies the specific elements responsible for the error.

In the provided example, the string value "14" [representing the price of fruits in the south region] is identified as one

of the elements causing the error. Based on this insight, the program removes these unwanted elements while retaining the relevant ones. As a result, the input data is updated to exclude unnecessary elements, thereby refining the dataset for more efficient debugging. The output of this step serves as the basis for further optimization and fine-tuning in subsequent phases of the program.

The ultimate outcome, presented in Figure 6, displays the result achieved at the concluding stage.

```
{"state": "NY", "prices": {"region_wise": [12,  13]}}
{"state": "NY", "prices": {"region_wise": [10,"14"]}}
```

Fig. 6. Reduced input after step 3

The resulting reduced input is notably more compact than the original input, featuring fewer keys. This outcome substantially simplifies the manual inspection process for developers, who can now efficiently locate the specific JSONs and keys contributing to the error. Although the presented example may not encompass an extensive set of original JSON lines, it serves as an illustrative demonstration. Consider the scenario where the input involves many JSONs, each containing many keys. Even under these circumstances, the ultimate output remains consistent – a concise and focused representation streamlining the debugging effort. Developers grappling with vast datasets can benefit from a concise overview highlighting the key elements contributing to the error.

IV. Related work

This section explores existing research that closely aligns with the objectives and scope of our project. The following research papers have been selected due to their relevance to the key themes and concepts addressed in our work: Research paper by Leitner et al.'s [1] work focuses on addressing the challenge of efficient unit test case minimization. The paper presents innovative techniques for minimizing unit test cases while retaining their effectiveness in identifying software defects. By analyzing their approach to optimize the test suite and reduce redundancy, we gain insights into strategies that enhance testing efficiency without compromising coverage. The survey by Shin Yoo and Mark Harman [2] offers a comprehensive overview of regression testing minimization, selection, and prioritization techniques. The paper delves into the methodologies used to optimize regression testing processes, emphasizing the importance of identifying high-impact test cases while minimizing redundant executions. By studying the survey's findings, we gain a broader understanding of strategies that enhance regression testing efficiency in software development. Yong Lei and J. H. Andrews explore the minimization of randomized unit test cases in their research [3]. The paper investigates techniques for reducing the number of generated random test cases while maintaining adequate test coverage. By delving into their approach to improve the efficiency of randomized testing, we gather insights that can enhance our understanding of minimizing test cases within randomized testing scenarios.

While the previously mentioned research primarily focuses on reducing the number of unit test cases, there is a notable gap concerning the consideration of input data size. This paper addresses this gap by emphasizing the input data aspect more than solely concentrating on the number of unit test cases. This nuanced perspective proves particularly beneficial for developers engaged with substantial data sets. In scenarios where issues arise within the input data, pinpointing the exact source of failure can be an intricate challenge. As such, this paper offers a comprehensive approach that provides essential insights into managing extensive and potentially complex data.

V. Conclusion

In this project, we have successfully developed an innovative approach to streamline the debugging process for developers working with large JSON lines (JSONL) data. Our program systematically reduces the size and number of JSON objects in the input data, providing developers with more manageable subsets for efficient debugging. The results of our approach have demonstrated its effectiveness in simplifying the identification and resolution of issues within large JSONL datasets. By providing developers with more focused datasets, our program reduces the time and effort required for manual debugging and enables a more systematic and reliable debugging process.

Throughout the project, we have analyzed various real-world scenarios and datasets, showcasing the adaptability and robustness of our approach in handling diverse use cases. Our implementation has shown promising results, successfully reducing the debugging complexity and improving the overall development workflow.

In conclusion, our project presents a valuable contribution to the development community by offering a practical and effective solution for handling large JSONL data and simplifying the debugging process. As the volume and complexity of data continue to grow, our approach offers a promising way to tackle the challenges associated with debugging and data analysis in modern software development environments.

VI. Future work

The project opens up several promising avenues for further exploration and enhancement. One noteworthy direction involves refining the program's subset selection process by leveraging intelligent algorithms. By strategically choosing subsets for evaluation, the reduction process could be optimized to a significant degree. Incorporating statistical or machine learning techniques could empower the program to dynamically select subsets with a higher likelihood of containing errors. This intelligent approach would streamline the reduction process and provide developers with more focused and informative subsets for effective debugging. Consequently, this could lead to accelerated error identification and resolution, ultimately improving the overall efficiency of the debugging process.

In addition, there is a compelling opportunity to enhance the program's performance by applying advanced optimization techniques. One such avenue is the implementation of parallel processing and memory optimization strategies. By harnessing the power of parallel computing, the program could efficiently distribute computational tasks, substantially reducing processing time for large JSON lines files. Furthermore, memory optimization techniques could be employed to manage and utilize system resources more efficiently, preventing memory-related bottlenecks and ensuring smoother program execution. By incorporating these performance optimization techniques, the program's scalability could be significantly enhanced, enabling it to handle more extensive datasets without compromising efficiency. This scalability improvement would cater to the needs of developers who regularly work with substantial data files, offering them a tool that remains effective and responsive even when dealing with large volumes of information.

ACKNOWLEDGMENT

I extend my heartfelt gratitude to Dr. Michael Mior, my dedicated capstone advisor, for his invaluable guidance, unwavering support, and insightful mentorship throughout the duration of this project. My sincere thanks extend to my peers and colleagues who contributed to fruitful discussions and provided valuable insights that enriched the development and refinement of this project. Furthermore, I am grateful to my family and friends for their unwavering support, understanding, and motivation throughout this endeavor. Their encouragement has been a constant source of inspiration.

REFERENCES

[1] A. Leitner and M. Oriol. (2007, Nov.) Efficient unit test case minimization. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/1321631.1321698

[2] S. Yoo and M. Harman. (2012, Mar.) Regression testing minimization, selection and prioritisation: A survey. software testing, verification, and reliability. [Online]. Available: https://doi.org/10.1002/stvr.430

[3] Y. Lei and J. H. Andrews. (2005, Nov.) Minimization of randomized unit test cases. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1544741