Developing a Comprehensive Taxonomy for REST API Faults: Insights from Real-World Bug Reports

Aakash Kulkarni
School of EECS
Oregon State University
Corvallis, United States
kulkaraa@oregonstate.edu

Soon Song Cheok
School of EECS
Oregon State University
Corvallis, United States
cheoks@oregonstate.edu

Shreyes Joshi
School of EECS
Oregon State University
Corvallis, United States
joshish@oregonstate.edu

***Abstract*—The increasing reliance on REST APIs in modern web applications has highlighted the need for effective fault localization techniques. Existing taxonomies for REST API faults, primarily based on automated test generation tools, focus on error messages and status codes. However, these taxonomies often fail to account for real world issues that can occur in REST APIs, leading to a gap in fault classification. In this study, we analyze real-world bug reports from GitHub to develop a comprehensive taxonomy that encompasses a wider range of fault types. By examining 24 bug reports, we identified 12 distinct categories of faults that are not adequately represented in current taxonomies. Our findings indicate that combining test suite-based taxonomies with bug report-based taxonomies can bridge the gap, providing a more holistic understanding of faults in REST APIs. This new taxonomy aims to enhance fault localization, improve API robustness, and better align with real-world scenarios encountered by developers. Alongside this taxonomy, we created a benchmark for evaluating these 24 bugs, which are real world bugs.**

***Index Terms*—Fault Localization, REST APIs, Taxonomies**

I. Introduction

REST APIs (Representational State Transfer) have become backbone of the modern web and cloud applications. They facilitate seamless interactions between client and server through stateless communication, enabling services to be scalable, reliable, and easily integrateable [1] [2]. Basically, REST APIs are a set of rules and standards defined by OpenAPI [3] used to enable communication between different software applications over the internet. They are built around the use of standard HTTP methods such as GET, POST, PUT, and DELETE to interact with resources, which are any kind of data or service that can be named on a network. Given their critical role, the effective identification and resolution of faults within REST APIs remain a significant challenge [4], prompting the need for research on how well the existing taxonomy align with real-world scenarios.

As systems that rely on REST APIs grow in scale and complexity [5] , minor faults can escalate into major disruptions, impacting user experience and business operations. So it is important to understand the real-world faults, but there are many faults that can be existing in the context of REST APIs. And we need a detailed taxonomy to cluster these real-world bugs into meaningful categories. So, that we can prioritize and diagnose the critical faults in REST APIs.

Our initial assumption was that the existing taxonomy, developed from automated test generation tools, would align well with real-world bug reports from GitHub. To empirically test this assumption, we conducted a study analyzing a set of bug reports from various REST API projects on GitHub. Our findings revealed that the existing taxonomy failed to account for real world issues that can occur under successful response codes (e.g., 200 OK). Also, the paper ”On the Faults Found in REST APIs by Automated Test Geneation” [6] was limited to the test suite based results and they made a taxonomy based on those results. This significant misalignment between the test suite based taxonomy and with real world bugs highlighted the need for a new, comprehensive taxonomy.

Given the shortcomings of the existing taxonomy, our primary objective was to develop a new taxonomy that better represents the faults encountered in real-world scenarios. Alongside this taxonomy, we aimed to create a benchmark for evaluating these faults, providing a structured framework for assessing and improving REST API reliability.

And the combining the test-suite based taxonomy and the bug report-based taxonomy, we created a comprehensive taxonomy that encompasses a wider range of faults. This new taxonomy aims to provide a more holistic understanding of faults in REST APIs, improving API robustness, and better align with real-world scenarios.

Our Assumptions:

- *Assumption 1*: We assume that the findings and the new taxonomy can be generalized to other REST API projects beyond the ones specifically analyzed in this study.
- *Assumption 2*: We assume that the new taxonomy will provide better diagnostic capability for developers, allowing them to identify, categorize, and prioritize faults more effectively.

The motivation for this research stems from the critical need to address the gap between theoretical fault models derived from automated test generation tools and the practical issues encountered in real-world REST API applications. Existing taxonomies primarily focus on certain types of errors, such as 500-based errors, and often fail to account for faults that occur under successful response codes like 200 OK. This misalignment limits the ability of developers to effectively diagnose, categorize, and prioritize faults. By creating a new, comprehensive taxonomy based on real-world bug reports, we aim to provide a more accurate and practical framework for understanding REST API faults. This taxonomy will enhance fault understanding process, improve system reliability, and better support developers in maintaining robust and resilient REST API-based systems.

To guide our investigation and address the identified gaps, we formulated the following research questions:

1) **RQ1:** What categories of faults are most prevalent in real-world REST API bug reports?
2) **RQ2:** How does the new taxonomy improve our understanding and classification of REST API faults compared to the existing taxonomy?

For the evaluation dataset, we will manually select the issues from the GitHub services and their fixes from REST API projects that utilize Spring Boot or Jersey frameworks. These GitHub services are used in the paper *Generating REST API Specifications through Static Analysis* by Manish et al [7] This dataset will encompass various categories of faults identified in REST APIs. By analyzing repositories and commit histories, we will extract specific instances where bugs have been documented and subsequently fixed.

Our analysis of bug reports led us to create a benchmark of 24 bugs which were classified with a taxonomy based on bug reports.

In the following sections we will describe our methodology and results.

II. Background and Motivation

A. Importance of REST APIs

REST APIs (Representational State Transfer APIs) are central to modern web and cloud applications, serving as critical conduits for data exchange and system integration. They leverage standard HTTP methods like GET, POST, PUT, and DELETE to interact with networked resources, making them integral to the architecture of distributed systems. The scalability, reliability, and ease of integration provided by REST APIs facilitate seamless interactions between clients and servers, thereby enhancing the performance and flexibility of complex software ecosystems. Their widespread adoption underscores their importance in today's technology landscape, where rapid communication and data accessibility are paramount.

B. Gaps in Current Research

The unique operational dynamics and architectural complexities of REST APIs pose new challenges that are not fully addressed by existing to create a comprehensive taxonomy. Previous studies have developed fault taxonomies for REST APIs which are helpful only in categorizing based on the failed test cases.

III. Related Work

The paper titled "On the Faults Found in REST APIs by Automated Test Generation" [6] investigates the types of faults that can be uncovered through the use of automated test generation tools. This research provides a taxonomy of faults specifically found in REST APIs by analyzing the failures encountered during automated testing. The study meticulously categorizes various fault types based on the error messages and behavior observed when automated test cases are executed against REST API endpoints.

This is the only paper we found that attempts to create a comprehensive taxonomy of faults for REST API applications. However, its scope is limited to faults identified through test suites. The taxonomy developed in this research is based on the results of these automated tests, which primarily detect faults resulting in error status codes, such as 500 Internal Server Errors. Consequently, this taxonomy may not capture the full range of faults that occur in real-world scenarios, particularly those that manifest under successful response codes.

Furthermore, the classification process in this paper is heavily manual, involving significant human effort to categorize the faults based on their characteristics. Currently, there is no automated classification method that can effectively cluster these faults into meaningful categories. This reliance on manual classification poses challenges in scalability and consistency, as the subjective nature of human judgment can lead to variations in fault categorization.

Overall, while this research provides valuable insights and a foundational taxonomy for REST API faults, its limitations highlight the need for a more comprehensive and automated approach to fault classification.

IV. Approach

A. Dataset Creation and Preparation

In our research, we utilize a set of REST API services previously employed in the study "Generating REST API Specifications through Static Analysis" by Manish et al [7]. These services are chosen due to their diverse characteristics and well-documented configurations, making them suitable subjects for evaluating fault localization techniques.

We are three graduate students with knowledge on REST API, we have selected the issues on GitHub Repositories, and we have the read the bug reports of the issues and saw what were the changes made and assigned it a fault type. We all have gone through all the services and then we together consolidated into one single list where we found 24 Faults which we can test using the Fault Localization Techniques. There were almost 50 bugs we had discussed, and we are confident about these 24 bugs in our dataset. And the others are pruned for now because we are not confident enough about those.

B. Integration with REST API Characteristics

Given the unique challenges posed by REST APIs, such as stateless operations and the presence of non-code artifacts

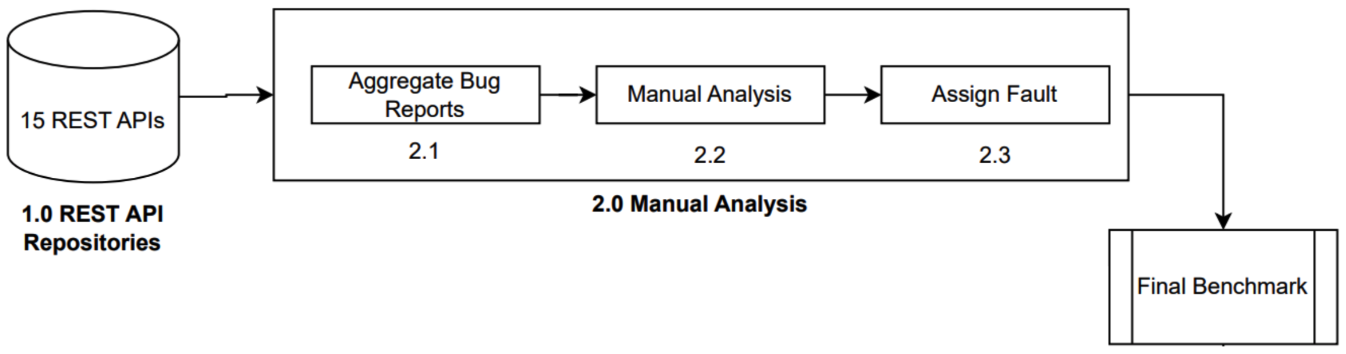

Fig. 1. Approach Overview

TABLE I
LIST OF REPOSITORIES

No.	Repository Name	# of Faults
1	management-api-for-apache-cassandra	2
2	cwa-verification-server	7
3	digdag	2
4	ohsome-api	10
5	kafka-rest	3

(e.g., configuration files and API specifications), our approach will also consider these elements:

- **Non-Code Artifacts**: Special attention will be given to faults that may originate from or affect non-code artifacts.

V. EVALUATION

A. Dataset

To conduct this research, a dataset is required for evaluation. The dataset contains REST API faults, their sources, and other relevant information necessary like Bug Reports to evaluate FL techniques.

The dataset we started with had GitHub APIs from 15 different REST API repositories from the services were management-api-for-apache-cassandra, catwatch, cwa-verification-server, digdag, enviroCar-server, features-service, gravitee-api-management, kafka-rest, ocvn, ohsome-api, proxyprint-kitchen, quartz-manager, restcountries, senzing-api-server, and Ur-Codebin-API. The process of filtering out issues not marked as 'bugs' and 'closed,' leaving only the set of issues that have been fixed details I. Issues involving code changes are collected and then further filtered through manual analysis to select those applicable to this research. These services provide us services with bug information we need to create a benchmark on REST API faults.

Our initial assumption that the existing taxonomy would be sufficient proved incorrect, prompting us to develop our own taxonomy. When we encountered issues not covered by the existing taxonomy, we created new categories. Out of the 24 bugs analyzed, only 2 could be classified using the existing taxonomy. However, even for these, the alignment between the bug reports and the fault type definitions was not ideal.

B. The Taxonomy

Based on our analysis of the 24 bug reports from various REST API projects, we developed a comprehensive taxonomy 2 that encompasses a broader range of fault types not adequately captured by existing taxonomies. This new taxonomy is categorized into major fault categories, each containing specific fault types.

Major Categories and Fault Types

C. Major Categories and Fault Types

1) **Concurrency Issues**
   - **Concurrency Issue:** This fault refers to a scenario known as a race condition in concurrent programming, where two functions are intended to execute sequentially but run simultaneously instead. This can result in one function depending on the output of the other function before it has completed, leading to erroneous outcomes or failures.
2) **Configuration Issues**
   - **Environment Incompatibility:** These are the kind of errors which arise due to the environments such as Operating System, Hardware Environment, or Network Environment, where the code written for one environment doesn't translate to another environment.
   - **Incorrect Configuration:** This fault refers to errors or mistakes in the setup or configuration of a system, software, or application. Misconfiguration can lead to unexpected behavior, errors, or issues.
3) **Implementation Issues**
   - **Coding Issues:**
     - **Hard-coded:** This fault type occurs when a value is invalid due to being Hard-Coded within the software.

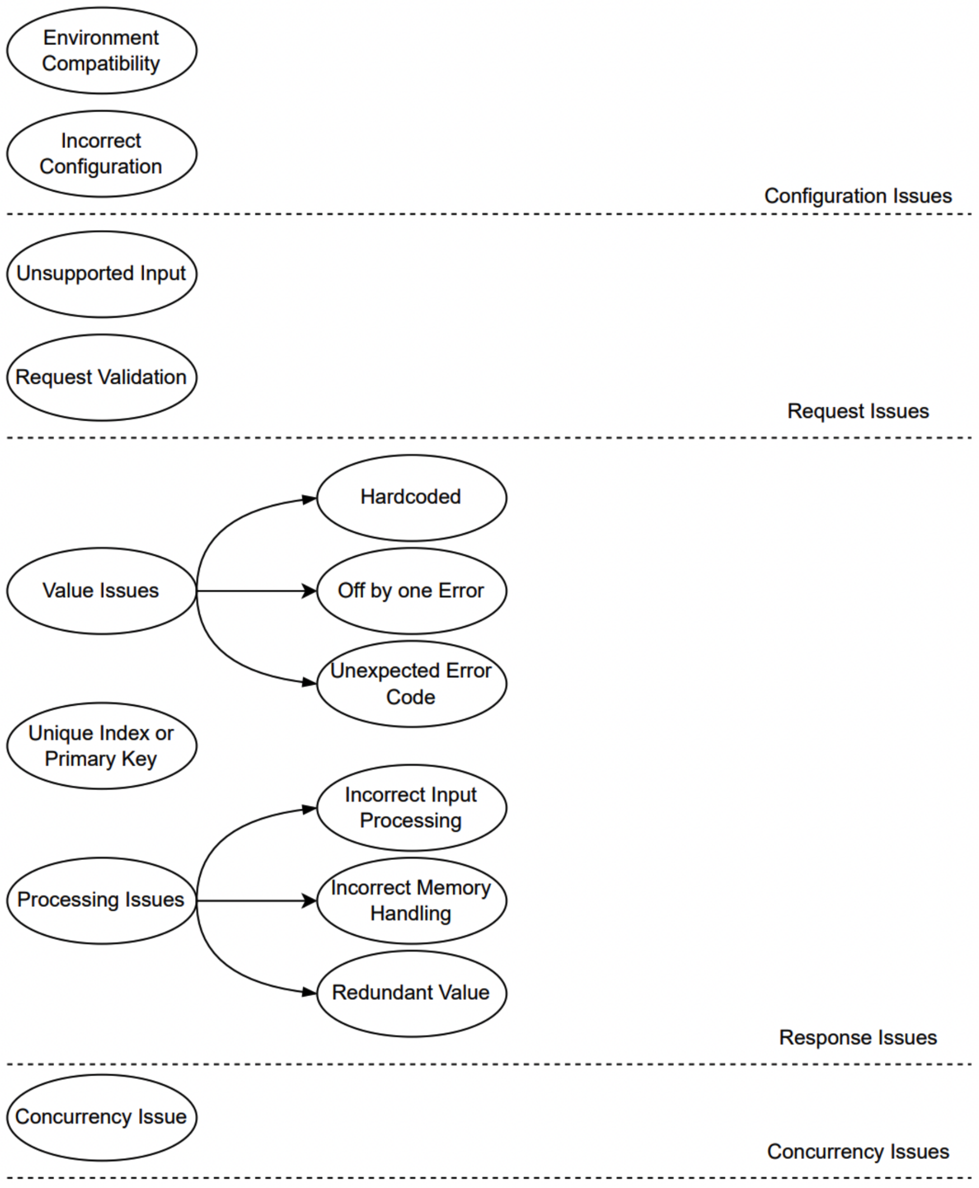

Fig. 2. Developed Taxonomy

  - **Off by one Error:** An off-by-one error refers to a programming mistake where the application incorrectly processes or manipulates data, resulting in either accessing one more or one less element than intended, leading to unintended behavior or functional errors in the web application.
  - **Unsupported Input:** This issue occurs when the documented functionality or parameters are not implemented or supported in the actual codebase or system. This discrepancy can lead to unexpected behaviors, errors, or limitations when utilizing the software based on the documented specifications.
- **Processing Issues:**
  - **Incorrect Input Processing:** Incorrect input processing in web applications refers to the improper handling of user inputs, leading to issues such as errors in API responses, unexpected behavior, and inconsistencies in the way data is processed and displayed. This can result in inaccuracies, unexpected outcomes, and functionality failures within the web application.
  - **Incorrect Memory Handling:** Incorrect memory handling involves inadequate or absent measures to limit or manage memory usage within a system, potentially leading to issues like memory exhaustion, performance degradation, or system instability.

- **Redundant Value:** This fault type refers to an unnecessary or duplicated data point or condition within a system, schema, or dataset. It indicates an extra check or condition that adds complexity without providing any additional benefit in terms of data validation or processing logic.

- **Indexing Issues:**
  - **Unique Index or Primary Key:** This fault arises when an identifier, such as an ID or key, is shared by multiple entities or records. This lack of uniqueness can result in data inconsistency, referencing errors, and challenges in accurately identifying or distinguishing individual entities within the system.

4) **Verification and Validation Issues**
   - **Request Verification:** This refers to the process of confirming the validity and authorization of incoming requests made to a system or application. This involves checking user permissions, ensuring proper authentication, and validating the data provided in the requests to prevent unauthorized access, maintain security, and uphold data integrity within the system.
   - **Unexpected Error Code:** This occurs when an error code is generated in response to a specific condition that deviates from the expected or standard error code for that situation.

D. Results

RQ1: What categories of faults are most prevalent in real-world REST API bug reports?

Based on our analysis of 24 bug reports from various REST API projects, we identified several prevalent categories of faults. The most common fault categories included:

TABLE II
FAULT DATA

No.	Fault Type	# of Faults
1	Concurrency Issue	2
2	Environment Incompatibility	1
3	Hard-Coded	1
4	Incorrect Configuration	3
5	Incorrect Input Processing	4
6	Incorrect Memory Handling	1
7	Off by one error	1
8	Redundant Value	1
9	Regression Failure	1
10	Request Verification	2
11	Unexpected Error Code	1
12	Unique Index or Primary Key	2
13	Unsupported Input	4

Based on the table above, we identified 2 categories of faults that are most prevalent in real-world REST API bug reports. These include: Incorrect Input Processing and Unsupported Input.

1) **Incorrect Input Processing:** This fault type involves improper handling of user inputs, which can lead to errors in API responses, unexpected behavior, and inconsistencies in data processing. Such issues are critical as they directly impact the functionality and reliability of the API, making it essential for developers to implement robust input validation and processing mechanisms.
2) **Unsupported Input:** These faults occur when the API does not handle certain input types or values as expected, leading to failures or incorrect responses. This highlights the need for comprehensive input handling and testing to ensure that the API can gracefully manage a wide range of input scenarios, thereby improving its robustness and user experience.

RQ1

These findings emphasize the importance of focusing on input handling mechanisms to enhance the reliability and robustness of REST APIs. By addressing these prevalent fault types, developers can significantly reduce the occurrence of common issues, leading to more stable and dependable API services.

RQ2: How does the new taxonomy improve our understanding and classification of REST API faults compared to the existing taxonomy?

The new taxonomy provides several improvements over the existing taxonomy derived from automated test generation tools:

1) **Comprehensive Coverage:** The new taxonomy encompasses a broader range of fault types, including those that occur under successful response codes (e.g., 200 OK), which were not adequately captured by the existing taxonomy. This ensures a more comprehensive understanding of the various issues that can affect REST APIs.
2) **Real-World Relevance:** By analyzing real-world bug reports from GitHub, the new taxonomy reflects the practical issues encountered by developers. This empirical basis makes the taxonomy more relevant and useful for diagnosing and addressing faults in real-world scenarios.
3) **Detailed Categorization:** The new taxonomy includes specific subcategories for different types of implementation issues, processing issues, and configuration issues. This detailed categorization helps in pinpointing the exact nature of faults, facilitating more precise diagnosis and resolution.
4) **Improved Diagnostic Capability:** With a more accurate and detailed classification of faults, developers can better prioritize and address critical issues. The new taxonomy supports more effective fault localization and debugging processes, enhancing the reliability and robustness of REST API systems.
5) **Benchmark Creation:** The creation of a benchmark alongside the new taxonomy provides a structured framework for evaluating fault localization techniques.

This benchmark enables consistent assessment and comparison of different methods, driving improvements in fault detection and resolution.

RQ2

Overall, the new taxonomy significantly enhances our understanding and classification of REST API faults, bridging the gap between theoretical models (test suite based) and real-world scenarios, and providing practical tools for developers to improve API reliability.

VI. Discussion and Threats to Validity

A. Discussion

The development of a new taxonomy for REST API faults based on real-world bug reports provides a more comprehensive framework for understanding and classifying these faults. This taxonomy bridges the gap between theoretical models derived from automated test generation tools and the practical issues encountered in actual use. By encompassing a broader range of fault types, the new taxonomy offers several key benefits:

- **Enhanced Fault Diagnosis:** The detailed categorization of faults allows for more precise diagnosis and resolution, helping developers to identify and address critical issues more effectively.
- **Improved Fault Localization:** By providing a structured framework for evaluating fault localization techniques, the new taxonomy supports more effective fault detection and debugging processes.
- **Increased Relevance:** The empirical basis of the taxonomy, derived from real-world bug reports, ensures its relevance and applicability to the practical issues faced by developers.
- **Benchmark Creation:** The creation of a benchmark alongside the new taxonomy provides a valuable tool for assessing and comparing different fault localization methods, driving improvements in fault detection and resolution.

B. Threats to Validity

Despite the strengths of our study, there are several potential threats to validity that must be considered:

- **Selection Bias:** The bug reports analyzed in this study were selected from a specific set of GitHub repositories. This selection may not be representative of all REST API projects, potentially limiting the generalizability of our findings.
- **Manual Classification:** The process of categorizing faults was conducted manually, which may introduce subjective bias. Different researchers might categorize the same fault differently, affecting the consistency and reliability of the taxonomy.
- **Sample Size:** The study was based on 24 bug reports, which, although providing valuable insights, may not capture the full spectrum of possible faults in REST APIs. A larger sample size could provide a more comprehensive understanding.
- **Dynamic Nature of APIs:** REST APIs are continuously evolving, with new features and updates being added regularly. Our taxonomy might need to be periodically updated to remain relevant as APIs evolve.
- **Limited Automated Support:** Currently, the classification of faults into the new taxonomy is not automated, which could pose challenges in scalability and consistency for larger datasets.

Addressing these threats in future research could involve expanding the dataset, developing automated classification tools, and continually validating and updating the taxonomy to ensure its relevance and accuracy.

VII. Contributions

This research makes several significant contributions to the field of REST API fault localization and reliability:

- **New Taxonomy Development:** We developed a comprehensive taxonomy for REST API faults based on empirical analysis of real-world bug reports. This taxonomy captures a broader range of fault types, including those occurring under successful response codes, and provides detailed categorization for more precise fault diagnosis.
- **Empirical Validation:** By analyzing real-world bug reports from GitHub, we ensured that the new taxonomy reflects the practical issues encountered by developers, making it more relevant and useful for diagnosing and addressing faults in real-world scenarios.
- **Benchmark Creation:** We created a benchmark for evaluating the 24 bugs analyzed, providing a structured framework for assessing and improving fault localization techniques. This benchmark enables consistent assessment and comparison of different methods, driving improvements in fault detection and resolution.
- **Improved Fault Localization:** The new taxonomy supports more effective fault localization and debugging processes by offering a more accurate and detailed classification of faults. This enhances the reliability and robustness of REST API systems.
- **Tool for Developers:** Our research provides practical tools and insights for developers to better understand, categorize, and address REST API faults. By bridging the gap between theoretical models and real-world scenarios, we contribute to the development of more resilient and reliable API services.

These contributions collectively enhance our understanding and management of REST API faults, providing a foundation for future research and development in this critical area of software engineering.

VIII. Future Work

There are several avenues for future work to build upon the findings of this research and further enhance the understanding and management of REST API faults.

A. *Expansion of the Taxonomy*

Future research can focus on expanding the new taxonomy to cover a wider range of REST API projects and fault types. This can be achieved by:

- **Increasing Sample Size:** Analyzing a larger number of bug reports from a more diverse set of REST API projects to capture additional fault types and scenarios.
- **Continuous Updates:** Regularly updating the taxonomy to reflect new developments and emerging fault patterns in REST API technologies.
- **Automated Classification:** Developing automated tools for fault classification to improve scalability and consistency. These tools can use machine learning and natural language processing techniques to categorize faults based on their descriptions and characteristics.

B. *Testing Fault Localization Techniques*

Another important direction for future work is to evaluate the effectiveness of fault localization (FL) techniques, such as BLUES and LLMAO, using the benchmark created in this study. This can provide valuable insights into how well these techniques perform in the context of REST APIs. Specific steps include:

- **Empirical Evaluation:** Applying FL techniques like BLUES and LLMAO to the benchmark and assessing their performance using metrics such as Top-k accuracy and EXAM score.
- **Comparative Analysis:** Comparing the performance of different FL techniques to identify their strengths and weaknesses in localizing REST API faults.
- **Methodological Improvements:** Using the insights gained from empirical evaluations to refine and improve existing FL techniques, making them more effective for REST API fault localization.

By pursuing these future research directions, we can further enhance the robustness and reliability of REST API systems, ultimately contributing to the development of more resilient and dependable web and cloud applications.

References

[1] L. Li, W. Chou, W. Zhou, and M. Luo, "Design patterns and extensibility of rest api for networking applications," *IEEE Transactions on Network and Service Management*, vol. 13, pp. 154–167, 2016.

[2] A. Neumann, N. Laranjeiro, and J. Bernardino, "An analysis of public rest web service apis," *IEEE Transactions on Services Computing*, vol. 14, pp. 957–970, 2018.

[3] H. Ed-Douibi, J. Izquierdo, and J. Cabot, "Openapitouml: A tool to generate uml models from openapi definitions," in *International Conference on Web Engineering*. Cham, Switzerland: Springer, 2018, pp. 487–491.

[4] A. Barbir, C. Hobbs, E. Bertino, F. Hirsch, and L. Martino, "Challenges of testing web services and security in soa implementations," in *Test and Analysis of Web Services*. Berlin/Heidelberg, Germany: Springer, 2007, pp. 395–440.

[5] R. Khare and R. Taylor, "Extending the representational state transfer (rest) architectural style for decentralized systems," in *Proceedings of the 26th International Conference on Software Engineering*, 2004, pp. 428–437.

[6] B. Marculescu, M. Zhang, and A. Arcuri, "On the faults found in rest apis by automated test generation," *ACM Transactions on Software Engineering and Methodology*, vol. 31, pp. 1–43, 07 2022.

[7] R. Huang, M. Motwani, I. Martinez, and A. Orso, "Generating rest api specifications through static analysis," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3597503.3639137