CS 569 PROJECT PLAN REPORT

AGORA Revisited: A Replication Study on Innovative Test Oracle Generation for Advancing the Robustness of REST API

DONE BY

PARTH SHAH

SANKARANARAYANAN SRINIVASAN

HARSHINI REDDY KOUKUNTLA

TARUNI MOVVA

Introduction:

The rapid expansion of web services and the prevalence of microservices architecture have underscored the importance of Application Programming Interfaces (APIs) in contemporary software systems. Ensuring the reliability and quality of APIs is paramount, necessitating effective testing methodologies. AGORA, a groundbreaking approach introduced in a seminal research paper, promises to revolutionize API testing automation, offering improvements in efficiency and reliability. However, before integration into real-world development workflows, it is imperative to meticulously validate AGORA's claims and assess its performance under diverse conditions. This project proposes a comprehensive replication study aimed at investigating the scalability, efficiency, and effectiveness of AGORA, addressing the research questions in this report.

The Research Questions:

This replication study aims to address the following primary research questions:

a. How does the performance of AGORA scale with larger datasets of API requests and responses? Are there any bottlenecks in the system as the dataset size increases?

b. How does detecting and reducing duplicate input requests affect the performance of AGORA's API testing process?

c. How do different configurations of JSONMutator mutation operators affect the failure detection ratio achieved by AGORA?

Each research question delves into critical aspects of AGORA's functionality and its potential impact on API testing efficiency and reliability.

Motivation

The motivation behind this replication study is deeply rooted in the imperative need to rigorously validate and extend the findings presented in the original research on AGORA's capabilities in API testing automation. Addressing the three primary research questions is paramount due to their direct implications on the efficiency, scalability, and reliability of AGORA:

Scalability with Larger Datasets: Understanding how AGORA performs as the dataset size increases is crucial for its practical deployment in real-world scenarios. Scalability issues can hinder its effectiveness, potentially leading to bottlenecks and inefficiencies in testing workflows. By investigating AGORA's performance under varying dataset sizes, we aim to uncover any scalability limitations and identify strategies to address them, ensuring its applicability to diverse and large-scale API testing scenarios.

Impact of Duplicate Request Reduction: The presence of duplicate input requests can significantly impact AGORA's performance, potentially skewing performance metrics and hindering test coverage. By examining the effects of detecting and reducing duplicate requests, we aim to elucidate the extent to which AGORA's efficiency and effectiveness can be enhanced. Identifying and mitigating duplicate requests can lead to streamlined testing processes, improved resource utilization, and ultimately, more reliable API testing outcomes.

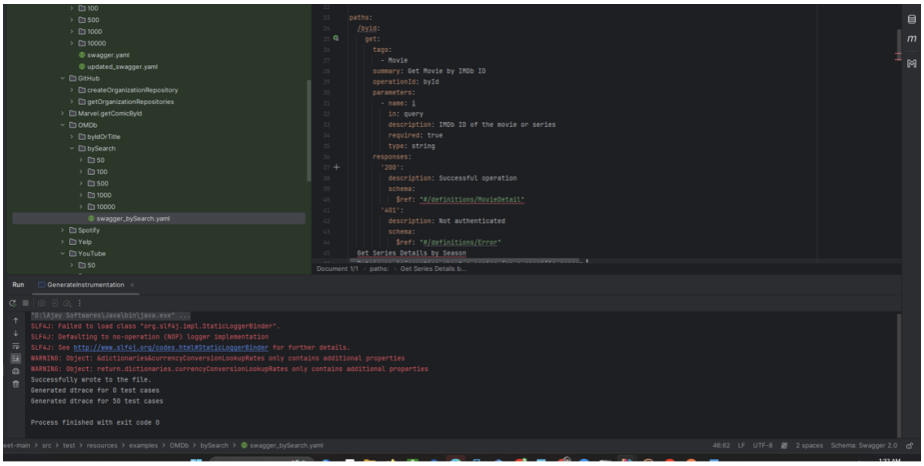
Effect of Mutation Operator Configurations: The selection and configuration of mutation operators play a pivotal role in AGORA's ability to detect failures accurately. By exploring different configurations of JSONMutator mutation operators, we aim to determine their impact on AGORA's failure detection ratio. Understanding which mutation operator configurations yield optimal results can inform developers and testers in fine-tuning AGORA for enhanced fault detection and improving overall testing efficacy.

In essence, this replication study seeks to provide empirical evidence and insights into AGORA's performance across various dimensions crucial for its practical utility in API testing. By addressing these research questions, we aim to validate AGORA's claims, identify potential challenges, and pave the way for its adoption as a reliable and efficient tool in the arsenal of software developers and testers. Ultimately, our endeavors aim to contribute to the advancement of automated API testing methodologies, fostering the development of more robust and resilient software systems.

Solution Approach and Exploration of Findings:

RQ1. How does the performance of AGORA scale with larger datasets of API requests and responses?

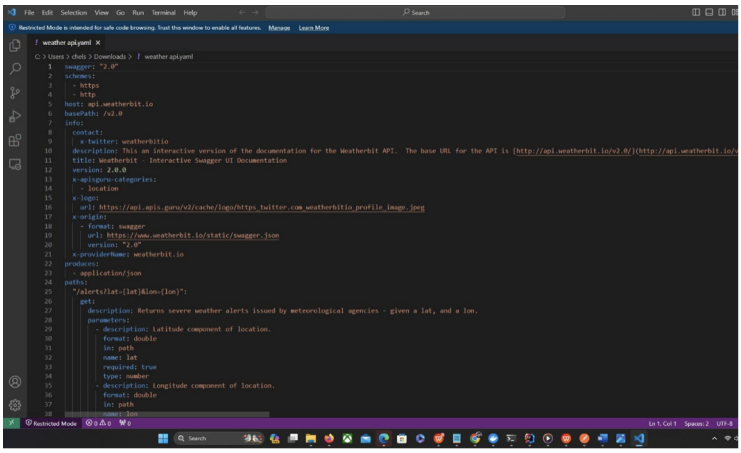
In our analysis of AGORA's scalability with enhanced datasets, we've developed two primary approaches. To evaluate AGORA's effectiveness with larger datasets, We expanded the OMDB dataset by adding new sources, such as the "Get Series Details by Season" option. We carefully selected these sources to ensure they work well with various data types, so as to check the efficiency of the AGORA working with data of many types. Below is an example of one of the sources we incorporated.



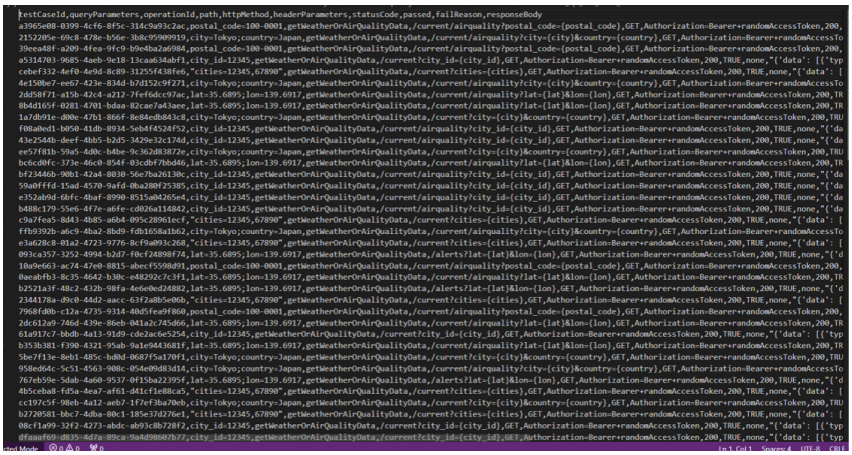
The API response for these endpoints are generated. Some of the response are manually are generated to make sure the API response are inline with the Open API format. The main point of this analysis is to ensure whether modified version of the daikon detects more invariant when compared with the invariant generated with existing dataset.

For our next step, we introduced a new endpoint by integrating the weather API, which we obtained from APIs.guru. We specifically selected this API because its endpoints provide numerical data. This choice was intentional to evaluate AGORA's capability in handling arithmetic comparisons. Additionally, we ensured to use the latest version of the API to guarantee that we are working with the most up-to-date responses.

Here is the OpenAPI spec for the Weather API



Here is the snapshot of the generated endpoint



We utilized a tool called POSTMAN for generating API responses. After generation, we systematically organized these responses into folders categorized by the size of the dataset, ensuring adherence to the existing organizational structure. To create the necessary .decl and .dtrace files for analysis, we employed BEET within the IntelliJ Integrated Development Environment (IDE).For the execution of Daikon, which is instrumental in identifying program invariants, we relied on Docker. This setup facilitated a streamlined process, enabling us to efficiently run Daikon with the generated files and ultimately produce valuable invariants.

Here is the example of the generated invariant for the weather API Response

1. /weather-api&getCurrentWeather&200&data():::EXIT;data.attributes.temp is Numeric;daikon.inv.unary.string.IsNumeric;(data.attributes.temp)
2. /weather-api&getCurrentWeather&200&data():::EXIT;data.attributes.weather one of {"Mild", "Sweltering", "Scorching", "Frost", "Cool", "Humid", "Rain", "Partly Cloudy", "Freezing", "Extreme Heat", "Muggy", "Chilly", "Breezy", "Clear", "Cloudy", "Wet"};daikon.inv.unary.scalar.OneOfScalar;(data.attributes.weather)
3. /weather-api&getCurrentWeather&200&data():::EXIT;data.type == "weather";daikon.inv.binary.twoString.StringEqual;(data.type)

Our primary objective was to assess the performance of the AGORA system using enhanced datasets. AGORA employs Dynamic Program Analysis, a method whose effectiveness is directly influenced by both the quality and quantity of the data it processes. Therefore, we are confident that as the size of the dataset expands, AGORA's efficiency—specifically its ability to identify more invariants—will also improve.

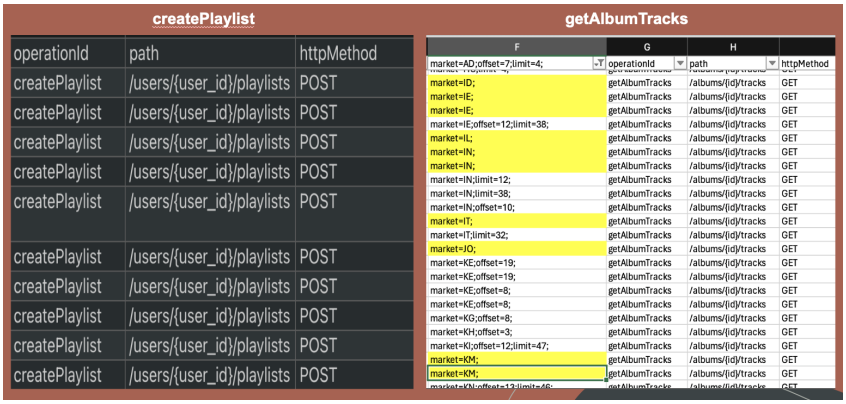
To measure the efficiency of AGORA for a larger dataset compared to the authors' existing dataset,

Efficiency = Number of relevant invariants generated from the extended dataset / Total number of invariants generated from the authors' dataset.

RQ2. Impact of Duplicate Request Reduction on AGORA's Performance

Observations:

During our preliminary analysis of the AGORA dataset, it became evident that duplicate input requests were prevalent. Specifically, we observed redundancy in the dataset, with certain API endpoints being called repeatedly with identical parameters. For instance, the "createPlaylist" endpoint exhibited duplicate requests with the same 'userid' parameter, while the "getAlbumTrack" endpoint consistently included the 'market' parameter across requests. Such redundancy in input requests can potentially skew performance metrics, leading to inefficiencies in testing efforts and resource utilization.



createPlaylist

operationId	path	httpMethod
createPlaylist	/users/{user_id}/playlists	POST
createPlaylist	/users/{user_id}/playlists	POST
createPlaylist	/users/{user_id}/playlists	POST
createPlaylist	/users/{user_id}/playlists	POST
createPlaylist	/users/{user_id}/playlists	POST
createPlaylist	/users/{user_id}/playlists	POST
createPlaylist	/users/{user_id}/playlists	POST
createPlaylist	/users/{user_id}/playlists	POST
createPlaylist	/users/{user_id}/playlists	POST
createPlaylist	/users/{user_id}/playlists	POST

Plan:

Experiment 1:

- Set a threshold value to control duplicate REST API requests in industrial REST API datasets.
- Limit duplicate records up to the threshold value.
- Provide the modified dataset CSV and OpenAPI Specification (OAS) to Beet as input.
- Use modified Daikon to produce invariants based on the Beet results.
- Apply JSONMutator tool to generate a test suite for the new dataset with limited duplicates.
- Detect any new invariants generated and evaluate the impact on the test suite.

Experiment 2:

- Repeat the steps from Experiment 1 with a different OAS specification, addressing another research question.
- Utilize a duplication removal utility to remove duplicates based on the threshold value.
- Record observations and compare results with Experiment 1.

Evaluation Metrics:

- Number of Unique Invariants Identified:

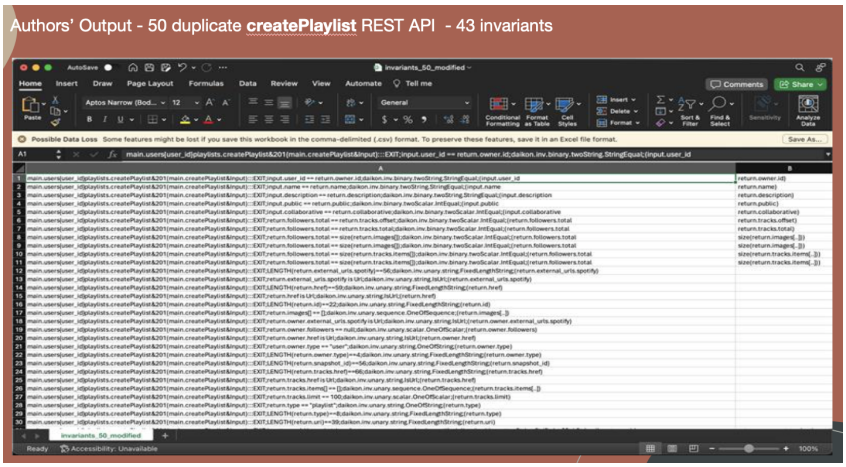
- This metric will assess the impact of removing duplicates on the diversity and richness of the invariants generated.
- A higher number of unique invariants could indicate more thorough test coverage.

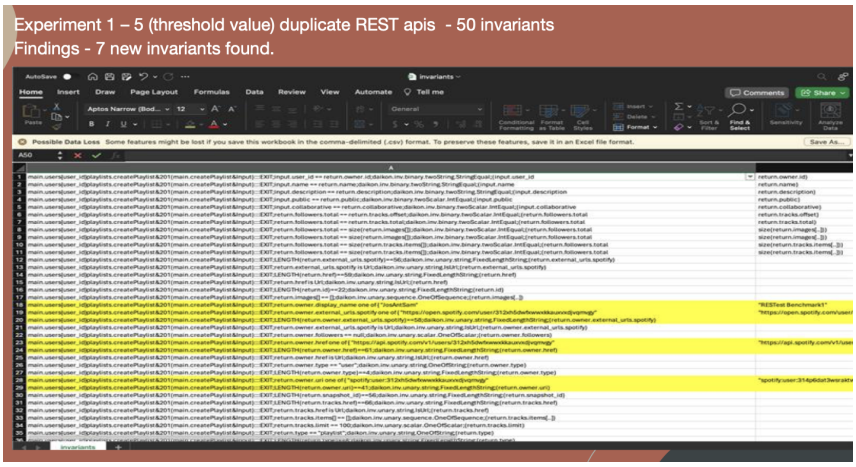
- Mutation Score:

- Apply mutation testing to the application and use the test suite to detect mutants.
- The mutation score, i.e., the percentage of mutants detected, can serve as a robust indicator of the quality of the test suite.

Investigating the impact of duplicate request reduction on AGORA's performance is crucial for understanding its efficiency and effectiveness in API testing. By conducting experiments and evaluating key metrics, we aim to provide insights into the significance of removing duplicates and its implications for test coverage and quality. These findings will contribute to enhancing the reliability and efficiency of AGORA as a tool for automated API testing.

Initial Findings - Experiment 1:





RQ3. How do different configurations of JSONMutator mutation operators affect the failure detection ratio achieved by AGORA?

JSONMutator is employed to introduce specific faults into the existing dataset deliberately. This is done to test whether the invariants identified by Daikon can successfully detect these introduced anomalies. With respect to the research question , We have decided to perform 3 experiments so as to explore the impact of the different factors on the invariant generation process and consequently on the failure detection capabilities of AGORA.

Updating OpenAPI Format with Existing Dataset

The OpenAPI Specification serves as a crucial guide for understanding an API's intended behavior. Changes in its structure or content are likely to affect the types of invariants that can be derived. In our investigation, we focused on the Amadeus Hotel API, comparing the most recent Swagger file obtained from APIs.guru with an earlier version. This comprehensive analysis revealed several key updates in the newer Swagger files. Here are the highlights:

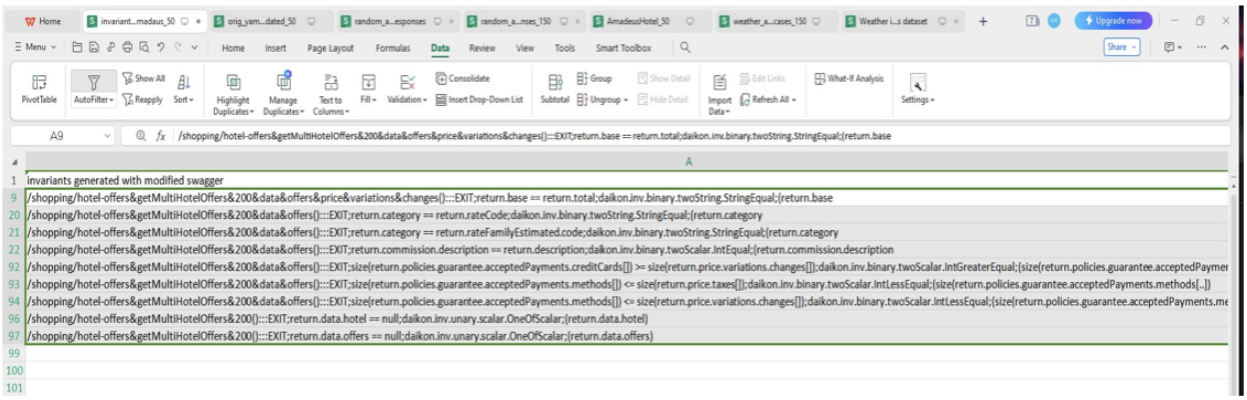
1. Version Update: The API version was updated from 3.0.5 to 3.0.8, indicating potential bug fixes or updates in API functionality.

2. Consistent Endpoints: Both versions feature the same endpoints, operating similarly, which shows a stable API functional range.
3. Parameter Adjustments: Minor modifications in parameters and their descriptions were noted, potentially impacting the generation of invariants.
4. Enhanced Enumerations: The latest version introduces expanded enum values, providing finer categorization of hotel offers.
5. Addition of New Properties and Descriptions: More detailed properties and descriptions have been incorporated into various definitions.

The API responses were intentionally left unchanged to evaluate how AGORA performs when API responses do not align with the OpenAPI specification.

The updated Swagger file, along with the existing test cases (100 Amadeus API), was processed through BEET, successfully producing trace and declaration files. These files were then analyzed by Daikon using a Docker command, resulting in the generation of invariants. The outcome revealed that the modified version produced fewer invariants compared to the original version, with Daikon identifying some unique invariants for the updated version. This discrepancy is primarily linked to changes in the parameters within the modified Swagger file.

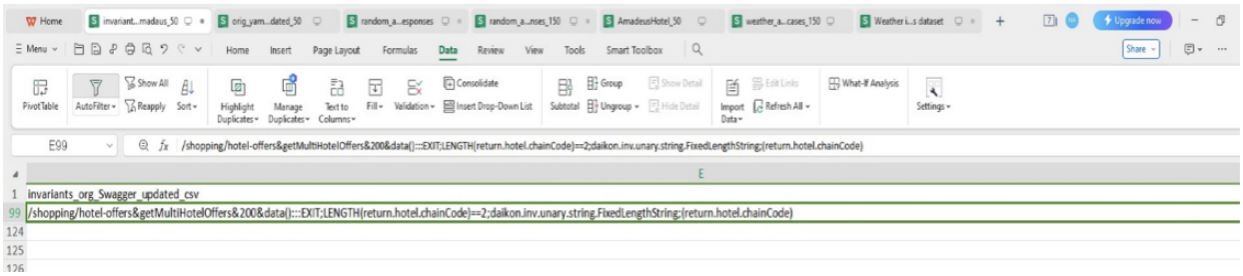
Here is the snapshot of the findings



Such a variation in the number of invariants directly impacts the Fault Detection Ratio, as a reduced count of invariants could lead to some faults going undetected.

2nd Experiment Updating API Responses with Previous Version of OpenAPI Format

In this part of the study, we examined the impact of changes in API responses, while keeping the OpenAPI specification version constant, on the detection of invariants. We manually altered the parameter values and removed some parameters in the response field of the Amadeus Hotel API. Subsequently, invariants were generated using Daikon. A comparison between the invariants from the original and modified versions revealed minimal differences; specifically, there was only one invariant that differed in the modified version. From this analysis, we deduce that minor modifications in the API responses do not significantly alter the invariants generated. Consequently, such changes are unlikely to substantially affect the fault detection ratio.



3rd Experiment Time Based Mutation Analysis

The approach of Time-based Mutation Analysis focuses on mutations affecting the time-related data within API responses, an aspect crucial for APIs in various industries that often contain at least one time-based variable. This analysis becomes particularly important for events with time constraints.

Taking an HRMS API (Human Resource Management System) as an example, consider if the API responses include time-sensitive data covering scenarios like:

- Adjusting Timestamps: Simulating cases where time data is set too far into the future or past compared to the expected timeline.
- Modifying Durations and Intervals: Altering time spans, for instance, extending or shortening a reservation duration.
- Changing Expiration Dates: Assessing the system's response to data that is either already expired or will expire earlier or later than initially planned.

To implement time-based mutations, the following steps are necessary:

- Identifying Time-Sensitive Data: Adapt the mutation logic to pinpoint fields in JSON objects representing time data, either through naming conventions or by directly indicating target fields.
- Developing Custom Mutation Operators: Tailor or set up mutation operators within JSONMutator to aptly manipulate time-based data. This includes:
- Adjusting dates and times within certain limits through JSON Mutation Properties.
- Injecting specific time-related errors, such as scheduling an event to start after its end time.
- Mutation Testing with AGORA: Utilize the AGORA testing framework on mutated test cases featuring time-based discrepancies.

If executed correctly, AGORA should identify faults (time-based mutants) assuming the time-based invariants are accurately generated by Daikon

Conclusion

In conclusion, this replication study endeavors to provide comprehensive insights into the scalability, efficiency, and effectiveness of AGORA for API testing automation. By systematically addressing the research questions outlined in the original paper, this study aims to validate AGORA's claims and assess its suitability for real-world deployment. The findings of this study will serve as valuable guidance for software developers and testers, facilitating informed decision-making regarding the adoption of AGORA in their testing workflows. Furthermore, this replication study contributes to the advancement of automated API testing methodologies, fostering the development of more reliable and efficient software systems.