CS 569 FINAL PROJECT REPORT

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

DONE BY

PARTH SHAH

SANKARANARAYANAN SRINIVASAN

HARSHINI REDDY KOUKUNTLA

TARUNI MOVVA

Abstract:

This replication study presents a systematic replication study and dataset extension to evaluate the performance of AGORA, an automated approach for test oracle generation in REST APIs. Through meticulous experimentation, we aimed to assess AGORA's capabilities and limitations under various conditions and scenarios.

Our study involved replicating the methodology outlined in the original research paper, conducting experiments with diverse datasets, and exploring the impact of controlling duplicate API requests on AGORA's performance. We set out to address research questions regarding the scalability of AGORA with larger datasets and the effect of reducing duplicate input requests on its performance.

Results from our experiments revealed valuable insights into AGORA's behavior. While AGORA demonstrated promise in detecting invariants and identifying API errors, we observed variations in its performance across different datasets. Notably, controlling duplicate requests had a discernible impact on AGORA's performance, with certain threshold values yielding significant changes in the generated invariants.

Our study offers valuable insights into AGORA's performance, highlighting areas for improvement. By emphasizing thorough experimentation and dataset curation, we provide a basis for future research in automated API testing.

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 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 results (if it captures new invariants?) of AGORA's API testing process? (Data deduplication using a threshold value)

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

Statement of the Problem:

In today's digital landscape, the reliance on REST APIs has become ubiquitous across various industries and domains. With the increasing complexity and criticality of these APIs in modern software systems, ensuring their reliability, security, and compliance with specifications is of paramount importance. However, traditional approaches to API testing often struggle to adequately address the nuanced challenges posed by these APIs, particularly in terms of generating accurate and efficient test oracles. Thus, the problem we aim to tackle in this study revolves around the need to enhance the effectiveness and scalability of automated testing for REST APIs, with a specific focus on improving test oracle generation methodologies through the utilization of AGORA.

Motivation

The motivation behind our endeavor stems from the critical role that REST APIs play in modern software development and the inherent challenges associated with ensuring their robustness and reliability. It is also 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 two primary research questions is paramount due to their direct implications on the effectiveness, 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.

Placement in Context of Related Research

Our research sits within the broader landscape of automated testing for REST APIs, which has witnessed significant advancements in recent years. AGORA represents a novel contribution to this field, building upon existing research in automated test case generation and dynamic invariant detection techniques. By leveraging insights from previous studies and incorporating innovative approaches to API analysis and instrumentation, AGORA stands out as a promising solution for addressing the challenges of test oracle generation in the context of REST APIs.

Description of Solution

Our approach to addressing the problem involved carefully replicating the methodology outlined in the original research paper on AGORA. This included utilizing Beet for API instrumentation, customizing Daikon for invariant detection, and conducting a series of experiments to evaluate AGORA's precision, scalability, and effectiveness. Additionally, we explored the impact of dataset size and duplicate request reduction on AGORA's performance, aiming to validate our findings through thorough analysis and draw meaningful conclusions.

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

Problem-Solving Methodology:

- **Replication and Analysis of Existing Data Across Four Datasets:**

1) Transfer of Existing Files:

We transferred the existing files, including the input YAML file, API responses, output declaration file, and data trace file, into a designated folder labeled "existing."

2) Beet Execution and Update of Current Dataset:

Following the transfer, we executed Beet to conduct dynamic analysis on the existing dataset. Upon completion, we copied the updated API response file, YAML file, newly generated trace file, and declaration file into a freshly created folder named "current," serving as the updated dataset for comparison.

3) Comparison of Current and Existing Files:

A comparison ensued between the declaration files and data trace files residing within the "current" and "existing" folders. This analysis facilitated the identification of any disparities or modifications between the datasets.

4) Generation of Invariants for Existing Data:

Utilizing Daikon, we generated invariants for the existing data trace files and integrated them into the designated "existing" folder, enabling the identification of expected behavior patterns within the original dataset.

5) Generation of Invariants for Current Data:

Employing Daikon once more, we generated invariants for the current data trace files, incorporating them into the designated "current" folder. This facilitated a comparative analysis of the invariants between the original and updated datasets.

6) Comparison of Generated Invariants:

A comparison was conducted between the generated invariants for the "existing" and "current" datasets, aiming to discern any variations or discrepancies in the expected behavior patterns across the datasets.

7) Identification of Differences:

Thorough examination of the compared invariants enabled the identification of any differences or inconsistencies between the datasets, providing valuable insights into potential disparities.

8) Reporting of Findings:

Upon observation of differences between the datasets, we documented and reported our findings, ensuring transparency and facilitating informed decision-making in subsequent research endeavors.

- **Extension of Dataset:**

In addition to replicating the existing analysis, we extended the dataset for further evaluation.

1) Configuration and Execution of RESTest:

After reviewing the RESTest documentation, we configured the framework to generate an expanded dataset comprising 15,000 API responses, focusing on the same API selected for the existing analysis. This adjusted configuration ensured comprehensive coverage and representation of API behaviors within the extended dataset.

2) Saving Expanded Dataset Responses:

Subsequently, we saved the 15,000 API responses into a newly created folder, ensuring organized storage and accessibility throughout the dataset extension process.

3) Beet Execution and Dataset Update:

Executing Beet on the expanded dataset facilitated dynamic analysis, with the output saved within the same designated folder to maintain consistency and organization throughout the analysis process.

4) Daikon Analysis:

Daikon was utilized to analyze the output from the expanded dataset, providing insights into the expected behavior patterns within the extended dataset.

5) Comparative Analysis of Invariants:

A comparative analysis ensued, comparing the detected invariants within the expanded dataset with those obtained from previously analyzed datasets comprising 10,000 and 50 responses, respectively.

6) Reporting of Findings:

Documentation and reporting of discernible differences in the detection of invariants across datasets of varying magnitudes.

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

- Objective:

The objective of this experiment was to assess the impact of controlling duplicate REST API requests on the generation of invariants using AGORA.

- Execution:

Threshold Value Set: A threshold value of 5 was established to control duplicate REST API requests within industrial REST API datasets.

The decision to set the threshold value to 5 was based on iterative testing and observation of AGORA's performance across various threshold values. Initially, higher threshold values, such as 10, were tested, but upon gradual reduction, it was observed that when the threshold reached 5, significant changes occurred in the invariants generated, particularly within the Spotify dataset.

This iterative approach allowed us to systematically explore the impact of different threshold values on AGORA's invariant generation process. By starting with higher values and progressively reducing the threshold, we aimed to identify the point at which meaningful changes in invariant generation occurred.

The emergence of new invariants in the Spotify dataset at the threshold value of 5 signified a critical threshold point where controlling duplicate REST API requests had a discernible impact on AGORA's performance. This observation provided valuable insight into the sensitivity of AGORA's invariant generation process to threshold values and justified the selection of 5 as the threshold value for this experiment.

Duplicate Records Limitation: Measures were implemented to restrict the number of duplicate records in the dataset to adhere to the established threshold value.

Input Modification: The modified dataset in CSV format, along with the OpenAPI Specification (OAS), was provided to Beet, the tool responsible for API instrumentation.

Invariants Generation: The modified Daikon tool was utilized to produce invariants based on the results obtained from Beet after processing the modified dataset.

Observation and Comparison: Observations were recorded regarding the generated invariants, and a comparison was made with those obtained from unmodified datasets to discern any variations or improvements.

Evaluation

Through a systematic evaluation, we thoroughly assessed AGORA's performance under different conditions and scenarios and examined AGORA's performance with respect to duplicate input requests. This involved conducting experiments using various datasets and analyzing the results carefully to identify patterns and potential areas for improvement. Our evaluation primarily focused on quantitative analyses and statistical assessments to provide a comprehensive assessment of AGORA's capabilities and limitations.

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

Findings:

- Original Result:

In our study, we compared the existing data with the current dataset to ascertain the accuracy of the results documented in the original research paper. This comparison was conducted across three distinct APIs: Yelp, Marvel, and YouTube. Each API was assessed with varying dataset sizes of 100, 1000, and 10,000 API responses, respectively. Upon meticulous analysis, we observed differences in the identified invariants between the current and existing datasets. Notably, these disparities were attributed to variations in the operating environments under which the datasets were generated. Furthermore, it was discerned that differences in the configuration settings of key tools such as Daikon and Beet could significantly influence the outcome, resulting in a distinct set of invariants.

Here are the Results

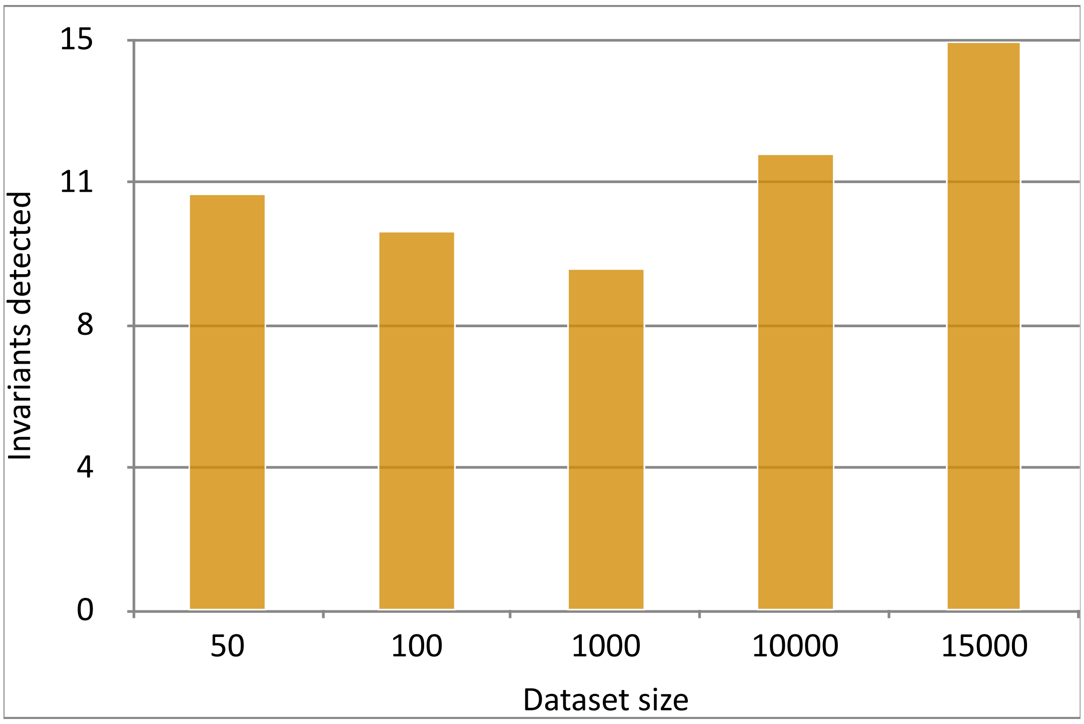
Existing_Daikon_modified_Results			
	100	1000	10000
Yelp	147	147	142
Marvel	133	112	102
Youtube	254	254	252

	Current_Modified_version_Results		
	100	1000	10000
Yelp	55	42	146
Marvel	109	95	94
Youtube	120	128	131

- Extension of Dataset Size:

To further explore the robustness of our analysis, we extended the dataset size, focusing on the OMDB API with its ByIdOrTitle endpoint. Leveraging a customized RESTest framework and Python code, we generated an expansive dataset comprising 15,000 API responses tailored to the existing Swagger file associated with the OMDB API endpoint. Careful consideration was given to crafting this dataset, ensuring its inclusivity of a broad spectrum of values, including edge cases.

For instance, we deliberately included scenarios such as the IMDb ID for the movie "Blade Runner," which has multiple versions, including the original 1982 release, as part of our edge case analysis.



Here is the table which represents the OMDB ByIdOrTitle invariants of different dataset sizes

The dataset size	50	100	1000	10000	15000
Invariants detected	11	10	9	12	15

Similarly, we meticulously adjusted the YouTube Data API YAML file and corresponding responses to evaluate potential variations in invariants. This assessment was specifically conducted with a dataset size of 10,000 API responses. The obtained results revealed noteworthy insights into the behavior of the API under varying dataset sizes.

Results:

(Youtube)	Current_Invariant	Modified_Invariant
The dataset size	10000	10000
Invariants detected	132	139

In our endeavor to expand API coverage, we introduced a new API named WeatherAPI, focusing on assessing the arithmetic aspects of API responses. For this purpose, a curated set of 150 API response records was generated. AGORA, our tool of choice for identifying invariants, successfully detected 8 invariants from these records. Notably, our testing phase aimed to scrutinize the generation of false positives by AGORA. Remarkably, during this evaluation, AGORA demonstrated robust performance by not producing any false positive invariants, underscoring its reliability in identifying true API behavior patterns.

Note : We faced problem with RESTest for the generation api request and responses since the API only accepted swagger file and not openAPI specs. We have also included the bug screenshot in the GitHub.

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

Controlling Duplicate REST API Requests

- Outcome:

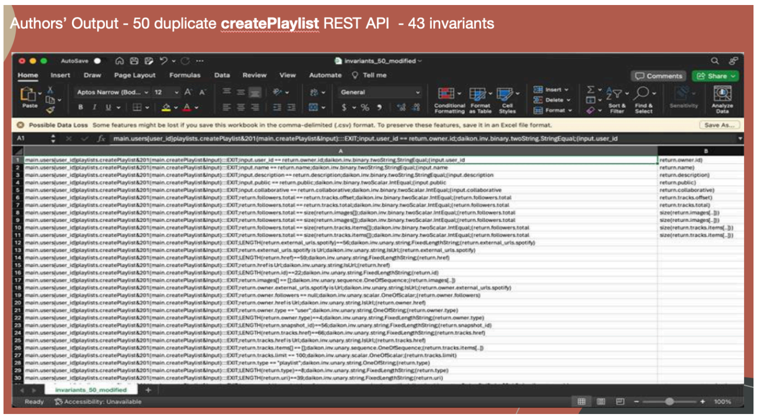
The dataset modification was successfully executed, ensuring adherence to the established threshold value.

Beet and Daikon processed the modified dataset, generating invariants as expected.

Upon observation and comparison, it was noted that while some endpoints produced a similar number of invariants after the deduplication process, others generated more invariants. This variation highlighted the nuanced impact of controlling duplicate REST API requests on AGORA's invariant generation process, indicating potential areas for further investigation and optimization.

However, it was observed that while the Spotify dataset produced new invariants after the deduplication process at the threshold value of 5, other datasets did not yield the desired results. In particular, some endpoints within other datasets either produced a similar number of invariants or showed no significant change. This discrepancy underscored the need for further investigation into the sensitivity of AGORA's performance to threshold values across different datasets.

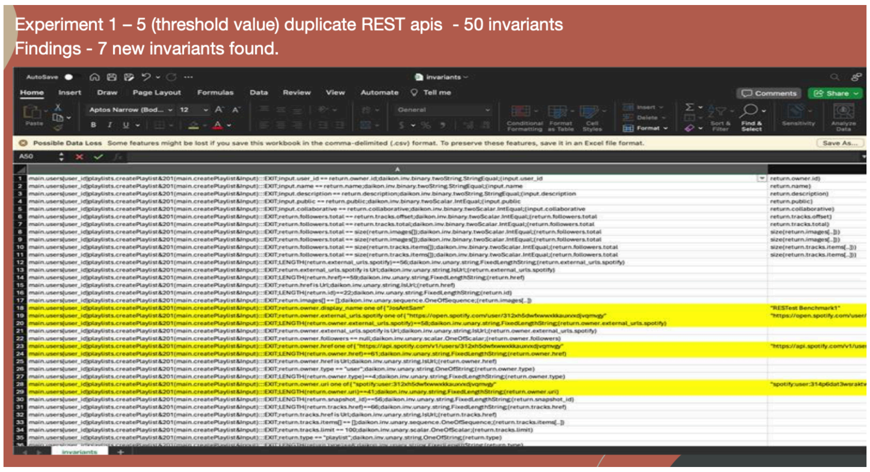
Findings from the Spotify dataset:


Authors' Output - 50 duplicate **createPlaylist** REST API - 43 invariants

invariants_50_modified
Possible Data Loss Some features might be lost if you save this workbook in the comma-delimited (.csv) format. To preserve these features, save it in an Excel file format.
Save As...
invariants_50_modified
Ready
Accessibility: Unavailable
100%

A1: main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;input.user_id == return.owner.id;daikon.inv.binary.twoString.StringEqual;(input.user_id

	A	B
1	main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;input.user_id == return.owner.id;daikon.inv.binary.twoString.StringEqual;(input.user_id	return.owner.id)
2	main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;input.name == return.name;daikon.inv.binary.twoString.StringEqual;(input.name	return.name)
3	main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;input.description == return.description;daikon.inv.binary.twoString.StringEqual;(input.description	return.description)
4	main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;input.public == return.public;daikon.inv.binary.twoScalar.IntEqual;(input.public	return.public)
5	main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;input.collaborative == return.collaborative;daikon.inv.binary.twoScalar.IntEqual;(input.collaborative	return.collaborative)
6	main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.followers.total == return.tracks.offset;daikon.inv.binary.twoScalar.IntEqual;(return.followers.total	return.tracks.offset)
7	main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.followers.total == return.tracks.total;daikon.inv.binary.twoScalar.IntEqual;(return.followers.total	return.tracks.total)
8	main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.followers.total == size(return.images[]);daikon.inv.binary.twoScalar.IntEqual;(return.followers.total	size(return.images[..]))
9	main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.followers.total == size(return.images[]);daikon.inv.binary.twoScalar.IntEqual;(return.followers.total	size(return.images[..]))
10	main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.followers.total == size(return.tracks.items[]);daikon.inv.binary.twoScalar.IntEqual;(return.followers.total	size(return.tracks.items[..]))
11	main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.followers.total == size(return.tracks.items[]);daikon.inv.binary.twoScalar.IntEqual;(return.followers.total	size(return.tracks.items[..]))
12	main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;LENGTH(return.external_urls.spotify)==56;daikon.inv.unary.string.FixedLengthString;(return.external_urls.spotify)	
13	main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.external_urls.spotify is Url;daikon.inv.unary.string.IsUrl;(return.external_urls.spotify)	
14	main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;LENGTH(return.href)==59;daikon.inv.unary.string.FixedLengthString;(return.href)	
15	main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.href is Url;daikon.inv.unary.string.IsUrl;(return.href)	
16	main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;LENGTH(return.id)==22;daikon.inv.unary.string.FixedLengthString;(return.id)	
17	main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.images[] == [];daikon.inv.unary.sequence.OneOfSequence;(return.images[..])	
18	main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.owner.external_urls.spotify is Url;daikon.inv.unary.string.IsUrl;(return.owner.external_urls.spotify)	
19	main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.owner.followers == null;daikon.inv.unary.scalar.OneOfScalar;(return.owner.followers)	
20	main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.owner.href is Url;daikon.inv.unary.string.IsUrl;(return.owner.href)	
21	main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.owner.type == "user";daikon.inv.unary.string.OneOfString;(return.owner.type)	
22	main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;LENGTH(return.owner.type)==4;daikon.inv.unary.string.FixedLengthString;(return.owner.type)	
23	main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;LENGTH(return.snapshot_id)==56;daikon.inv.unary.string.FixedLengthString;(return.snapshot_id)	
24	main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;LENGTH(return.tracks.href)==66;daikon.inv.unary.string.FixedLengthString;(return.tracks.href)	
25	main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.tracks.href is Url;daikon.inv.unary.string.IsUrl;(return.tracks.href)	
26	main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.tracks.items[] == [];daikon.inv.unary.sequence.OneOfSequence;(return.tracks.items[..])	
27	main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.tracks.limit == 100;daikon.inv.unary.scalar.OneOfScalar;(return.tracks.limit)	
28	main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.type == "playlist";daikon.inv.unary.string.OneOfString;(return.type)	
29	main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;LENGTH(return.type)==8;daikon.inv.unary.string.FixedLengthString;(return.type)	
30	main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;LENGTH(return.uri)==39;daikon.inv.unary.string.FixedLengthString;(return.uri)	



Experiment 1 – 5 (threshold value) duplicate REST apis - 50 invariants
Findings - 7 new invariants found.

```
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;input.user_id == return.owner.id;daikon.inv.binary.twoString.StringEqual;(input.user_id    return.owner.id)
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;input.name == return.name;daikon.inv.binary.twoString.StringEqual;(input.name    return.name)
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;input.description == return.description;daikon.inv.binary.twoString.StringEqual;(input.description    return.description)
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;input.public == return.public;daikon.inv.binary.twoScalar.IntEqual;(input.public    return.public)
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;input.collaborative == return.collaborative;daikon.inv.binary.twoScalar.IntEqual;(input.collaborative    return.collaborative)
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.followers.total == return.tracks.offset;daikon.inv.binary.twoScalar.IntEqual;(return.followers.total    return.tracks.offset)
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.followers.total == return.tracks.total;daikon.inv.binary.twoScalar.IntEqual;(return.followers.total    return.tracks.total)
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.followers.total == size(return.images[]);daikon.inv.binary.twoScalar.IntEqual;(return.followers.total    size(return.images[..]))
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.followers.total == size(return.images[]);daikon.inv.binary.twoScalar.IntEqual;(return.followers.total    size(return.images[..]))
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.followers.total == size(return.tracks.items[]);daikon.inv.binary.twoScalar.IntEqual;(return.followers.total    size(return.tracks.items[..]))
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.followers.total == size(return.tracks.items[]);daikon.inv.binary.twoScalar.IntEqual;(return.followers.total    size(return.tracks.items[..]))
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;LENGTH(return.external_urls.spotify)==56;daikon.inv.unary.string.FixedLengthString;(return.external_urls.spotify)
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.external_urls.spotify is Url;daikon.inv.unary.string.IsUrl;(return.external_urls.spotify)
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;LENGTH(return.href)==58;daikon.inv.unary.string.FixedLengthString;(return.href)
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.href is Url;daikon.inv.unary.string.IsUrl;(return.href)
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;LENGTH(return.id)==22;daikon.inv.unary.string.FixedLengthString;(return.id)
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.images[] == [];daikon.inv.unary.sequence.OneOfSequence;(return.images[..])
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.owner.display_name one of { "JosAntSam"    "RESTest Benchmark1"
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.owner.external_urls.spotify one of { "https://open.spotify.com/user/312xh5dwfxwwxkkauxvxdjvqmvgy"    "https://open.spotify.com/user/
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;LENGTH(return.owner.external_urls.spotify)==58;daikon.inv.unary.string.FixedLengthString;(return.owner.external_urls.spotify)
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.owner.external_urls.spotify is Url;daikon.inv.unary.string.IsUrl;(return.owner.external_urls.spotify)
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.owner.followers == null;daikon.inv.unary.scalar.OneOfScalar;(return.owner.followers)
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.owner.href one of { "https://api.spotify.com/v1/users/312xh5dwfxwwxkkauxvxdjvqmvgy"    "https://api.spotify.com/v1/user
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;LENGTH(return.owner.href)==61;daikon.inv.unary.string.FixedLengthString;(return.owner.href)
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.owner.href is Url;daikon.inv.unary.string.IsUrl;(return.owner.href)
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.owner.type == "user";daikon.inv.unary.string.OneOfString;(return.owner.type)
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;LENGTH(return.owner.type)==4;daikon.inv.unary.string.FixedLengthString;(return.owner.type)
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.owner.uri one of { "spotify:user:312xh5dwfxwwxkkauxvxdjvqmvgy"    "spotify:user:314p6dat3wsraktv
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;LENGTH(return.owner.uri)==41;daikon.inv.unary.string.FixedLengthString;(return.owner.uri)
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;LENGTH(return.snapshot_id)==56;daikon.inv.unary.string.FixedLengthString;(return.snapshot_id)
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;LENGTH(return.tracks.href)==66;daikon.inv.unary.string.FixedLengthString;(return.tracks.href)
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.tracks.href is Url;daikon.inv.unary.string.IsUrl;(return.tracks.href)
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.tracks.items[] == [];daikon.inv.unary.sequence.OneOfSequence;(return.tracks.items[..])
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.tracks.limit == 100;daikon.inv.unary.scalar.OneOfScalar;(return.tracks.limit)
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;return.type == "playlist";daikon.inv.unary.string.OneOfString;(return.type)
main.users{user_id}playlists.createPlaylist&201(main.createPlaylist&Input):::EXIT;LENGTH(return.type)==8;daikon.inv.unary.string.FixedLengthString;(return.type)
```

Number of unique invariants identified

	Results for Existing datasets					Results for Deduplicated datasets				
	10k	1000	500	100	50	10k	1000	500	100	50
Endpoints	**Spotify**									
createPlaylist	41	41	41	41	41	**50**	**43**	**43**	**43**	**46**
getAlbumTracks	90	84	84	83	80	90	84	84	**86**	80
getArtistAlbums	92	84	84	84	95	**94**	84	84	84	95
	Amadeus Hotel									
getMultiHotelOffers	226	227	228	220	223	226	227	228	220	223
	Yelp									
getBusinesses	142	147	149	147	130	142	147	**150**	147	130
	Github									
createOrganizationRepository	210	210	210	210	210	210	210	**206**	210	210
getOrganizationRepositories	169	169	168	164	167	169	**168**	168	164	167
	OMDb									
byIdOrTitle	32	33	33	33	33	32	33	33	33	33
bySearch	28	25	25	20	22	28	25	25	20	22

Conclusion

Through our systematic replication study and dataset extension, we have provided a comprehensive evaluation of AGORA, an automated test oracle generation approach for REST APIs. Our investigation involved replicating the methodology outlined in the original research paper, conducting experiments with varying dataset sizes, and exploring the impact of controlling duplicate API requests on AGORA's performance.

AGORA consistently captures key API behaviors and reveals deeper, more specific patterns as dataset sizes increase, demonstrating its scalability and enhanced behavioral analysis capabilities. Larger datasets enable a more comprehensive understanding of API behavior by introducing additional invariants without increasing false positives, ensuring the precision of AGORA's test oracles.

In particular, our experiments revealed that while AGORA shows promise in detecting invariants and identifying API errors, its performance can vary depending on factors such as dataset composition and threshold values for controlling duplicate requests. Notably, the threshold value of 5 yielded significant changes in the generated invariants for the Spotify dataset but did not yield desired results for other datasets.

Moving forward, future research should continue to explore strategies to enhance AGORA's robustness across diverse API testing scenarios. By further refining its algorithms and considering a broader range of edge cases, we can improve AGORA's effectiveness in detecting API errors and providing reliable test oracles. Overall, our study contributes to the ongoing efforts in advancing automated API testing methodologies and highlights the importance of thorough experimentation and dataset curation in evaluating the performance of such tools.

Data Availability

Link to trello board:

https://trello.com/b/PwYf9m1i/agora-project-group-8

Link to GitHub repository:

https://github.com/parthcompengg/CS569-AGORA-ProjectGroup8