Using LLM for Mining and Testing Constraints in API Testing

Minh-Hieu Huynh
Katalon LLC
Ho Chi Minh City, Vietnam
minhhieu2214@gmail.com

Quoc-Tri Le
Katalon LLC
Ho Chi Minh City, Vietnam
lqtri691@gmail.com

Tien N. Nguyen
Computer Science Department
University of Texas at Dallas
Texas, USA
Tien.N.Nguyen@utdallas.edu

Vu Nguyen
University of Science, VNU-HCM
Katalon LLC
Ho Chi Minh City, Vietnam
nvu@fit.hcmus.edu.vn

CCS Concepts

• **Software and its engineering → Software testing and debugging**.

Keywords

Large Language Models, API Testing

ACM Reference Format:
Minh-Hieu Huynh, Quoc-Tri Le, Tien N. Nguyen, and Vu Nguyen. 2024. Using LLM for Mining and Testing Constraints in API Testing. In *39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24), October 27-November 1, 2024, Sacramento, CA, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3691620.3695341

1 Problem

Testing Representational State Transfer (REST) APIs is crucial for ensuring the reliability and performance of APIs, which are essential to modern web services. This testing process helps identify and resolve issues related to data exchange and integration with other systems. Among the various API testing techniques, black-box testing relies on the OpenAPI Specification (OAS) to generate test cases and data. However, current API test automation methods are primarily focused on status code [10] and schema validation [1]. Status code validation involves ensuring that each HTTP request returns a response with a status code, a three-digit integer that indicates the outcome of the request. Schema validation verifies the correctness of the response data by comparing it to the schema. This includes checking that all required properties are present and that data types of these properties align with the schema specified.

While status code and schema validation are effective in ensuring correct data representation and status verification, they may miss issues related to the logical correctness and validity of the response data. For instance, if an API request asks for a flight before January 1, 2025, but the response provides a flight after that date, this discrepancy would not be caught by simply validating the status code or schema. Such constraints are critical to maintain the logical integrity and accuracy of results returned from APIs, which is essential for ensuring reliable operations. Current RESTful API testing approaches are limited in addressing this aspect.

ASE '24, October 27-November 1, 2024, Sacramento, CA, USA

ACM ISBN 979-8-4007-1248-7/24/10
https://doi.org/10.1145/3691620.3695341

2 Approach

We introduce APITesting, an approach designed to mine constraints from response bodies and generate test cases to validate these constraints. To achieve this, we utilize the capabilities of large language models (LLMs) to interpret natural language descriptions found in API specifications. These specifications often contain constraints on response bodies and operations, with the schema providing constraints on properties and descriptions of the returned object as a whole. Furthermore, we leverage LLMs' proficiency in generating source code to automatically create test cases based on the mined constraints, ensuring that the system under test correctly returns content that adheres to these constraints.

Our approach involves two steps: Constraints Mining and Constraints Test Generation. First, we use the OpenAPI/Swagger Specification as input for *Constraints Mining*, where we extract constraints from the natural language descriptions within the OAS file. Since constraint descriptions can appear in various sections of the OAS, we start by applying heuristics to extract descriptions related to operations, parameters, and response properties. We then employ an LLM to identify constraints from these descriptions. Second, the identified constraints serve as input for the LLM to perform *Test Case Generation*, producing code snippets that test these constraints. Each recognized constraint is paired with a snippet to verify that the response data aligns with the specified constraints.

Constraints Mining. To validate the response body of an API endpoint, we note that the API specification typically consists of two main components: the *request specification* (the API's *input*) and the *response schema specification* (the API's *output*).

1. The request specification outlines how to call the API endpoint, detailing the required *input parameters*, their roles, and the parameters within the request body, along with their respective functions. It may also include a description of the API's *operations*.

2. The response schema specification provides guidelines on the *response data*, describing each property in the response, its datatype, nullability, and other relevant details.

For constraints from response specifications, we first examine the descriptions within the response schema specification, as these descriptions provide the most straightforward constraints by directly mapping each property. In an OpenAPI specification, each endpoint specification contains a response schema specification that instructs clients on how to parse the data returned by the server in response. This response data specification structures the response object through its properties, including their descriptions.

Constraint Test Generation. Constraint tests are generated using an LLM and are tailored to two types of constraints identified earlier: Request-Response constraints and Response Property constraints. Request-Response constraints involve conditions based on the request parameter descriptions, while Response Property constraints are directly derived from the response schema's description.

These tests evaluate a response body alongside the request details and produce an outcome categorized as 'matched,' 'mismatched,' or 'unknown.' A 'matched' outcome indicates that the response body complies with the identified constraint. A 'mismatched' outcome indicates that the response body fails to meet the constraint. An 'unknown' outcome suggests that the response data does not include the property associated with the constraint, potentially because this property is optional in the specification.

1) *Request-Response Constraints Testing*

The Request-Response Constraint Test involves identifying a dependency between a constrained request parameter and the corresponding response property that reflects this constraint. To generate a test case for this scenario, we use a prompt to the LLM. This prompt requires four inputs: the parameter's name, the description of the parameter constraint, the corresponding property's name, and the response data schema.

We task the LLM with generating a validation function that accepts two parameters: *the response body* and *the request parameter*. The LLM's job is to create a script that checks the conditions between these two inputs. For example, to validate a 'created' time interval, the LLM needs to extract the 'created' time from the response body and the conditional values from the request parameter, such as 'created[gte]' (greater than or equal to) and 'created[lte]' (less than or equal to). It should then generate a logical comparison between these values.

2) *Response Property Constraints Testing*

The Response Property Constraint Test directly correlates the property's description with the property itself. To create a test case for this constraint, we utilize a specific prompt. This prompt requires three inputs: the property name, the constraint description, and the response data schema. The description extracted from the previous mining step gives the necessary information for generating constraint verification code, while the response data schema defines the structure and type of the expected data. This schema guides LLM in generating code to parse response data.

3 Related Work

Analysis of recent surveys on API testing [5, 7] reveals a clear trend towards automation adoption. Advancements in AI and ML have enhanced various aspects of API testing including of generating robust test cases [10], realistic test inputs [2], and to identify potential defects early in the development cycle [3].

KAT [8] leverages the language understanding capabilities of GPT to fully automate the API testing process using only an input OpenAPI specification. Kim *et al.* [6] applied GPT to augment OpenAPI specifications, enriching them with explanations of rules and example inputs generated by LLMs.

ARTE [2] aimed to generate test inputs for API testing, employing NLP, search-based, and knowledge extraction. Morest [9] is a model-based RESTful API testing method using a dynamically updating RESTful-service Property Graph. RESTler [4] is an automatic tool for stateful fuzzing of REST APIs, which analyzed OpenAPI specifications, inferred dependencies among request types, and dynamically generated tests guided by feedback from service responses using a test-generation grammar. RestTestGen [10] applied OpenAPI specifications for generating test cases, employing heuristics to ensure the robustness of the generated code by checking both response status codes and response data schemas.

4 Conclusion

This paper presents APITesting, a method for mining constraints from API response bodies and generating test cases to validate these constraints. By leveraging LLMs to understand descriptions in API specifications, APITesting generates test cases. Empirical results show that it achieves an average precision of 94.3% in constraint mining and up to 88.5% in test generation for constraint validation.

Acknowledgments

We thank Katalon inc. for sponsoring this research. Additionally, Tien N. Nguyen is supported in part by the US NSF grant CNS-2120386 and the NSA grant NCAE-C-002-2021, and Vu Nguyen is partially funded by the Faculty of Information Technology, University of Science, VNU-HCM and the Vingroup Innovation Foundation (VINIF) under the grant number VINIF.2021.JM01.N2.

References

[1] 2024. Postman API Testing Tool. https://www.postman.com/
[2] Juan C Alonso, Alberto Martin-Lopez, Sergio Segura, Jose Maria Garcia, and Antonio Ruiz-Cortes. 2022. ARTE: Automated Generation of Realistic Test Inputs for Web APIs. *IEEE Transactions on Software Engineering* 49, 1 (2022), 348–363.
[3] Andrea Arcuri and Juan P Galeotti. 2021. Enhancing search-based testing with testability transformations for existing APIs. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 1 (2021), 1–34.
[4] Vaggelis Atlidakis, Patrice Godefroid, and Marina Polishchuk. 2019. Restler: Stateful rest api fuzzing. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 748–758.
[5] Amid Golmohammadi, Man Zhang, and Andrea Arcuri. 2023. Testing RESTful APIs: A Survey. *ACM Trans. Softw. Eng. Methodol.* 33, 1, Article 27 (nov 2023), 41 pages. https://doi.org/10.1145/3617175
[6] Myeongsoo Kim, Tyler Stennett, Dhruv Shah, Saurabh Sinha, and Alessandro Orso. 2024. Leveraging large language models to improve REST API testing. In *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*. 37–41.
[7] Myeongsoo Kim, Qi Xin, Saurabh Sinha, and Alessandro Orso. 2022. Automated test generation for rest apis: No time to rest yet. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*. 289–301.
[8] Tri Le, Thien Tran, Duy Cao, Vy Le, Vu Nguyen, and Tien N. Nguyen. 2024. KAT: Dependency-aware Automated API Testing with Large Language Models. In *2024 IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE.
[9] Yi Liu, Yuekang Li, Gelei Deng, Yang Liu, Ruiyuan Wan, Runchao Wu, Dandan Ji, Shiheng Xu, and Minli Bao. 2022. Morest: model-based RESTful API testing with execution feedback. In *Proceedings of the 44th International Conference on Software Engineering*. 1406–1417.
[10] Emanuele Viglianisi, Michael Dallago, and Mariano Ceccato. 2020. Resttestgen: automated black-box testing of restful apis. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 142–152.