Google Scholar Crossref doi scopus

Impact factor 6.2

Geoscience Journal

ISSN:1000-8527

Indexing:

- Scopus
- Google Scholar
- DOI, Zenodo
- Open Access

www.geoscience.ac

Registered

AutoRestTest – HSL: A Hybrid Symbolic-Learning Guided Extension for REST API Testing

Mariya Jebastin P[1], Rajeswari R[2], Mohammed Jarshith J[3], and Bikramjit Thokchom[4]

[1] Department of Computer Applications, Sathyabama Institute of Science and Technology, Chennai 600119, INDIA

[2] Department of Computer Applications, Sathyabama Institute of Science and Technology, Chennai 600119, INDIA

[3] Department of Computer Applications, Sathyabama Institute of Science and Technology, Chennai 600119, INDIA

[4] Department of Computer Applications, Sathyabama Institute of Science and Technology, Chennai 600119, INDIA

Abstract—REST API testing is challenging since there is a tremendous number of combinations of operations, parameters, and dependencies, usually resulting in low coverage of the code and unnoticed faults. In this paper, a Semantic Property Dependency Graph (SPDG), an automated tool called AutoRestTest-HSL is presented, based on a novel algorithm: Hybrid Symbolic-Learning Guided Multi-agent Testing (HSL-MAT). The HSL-MAT algorithm is a multi-agent reinforcement algorithm based on symbolic reasoning on API constraints, where the agents are enabled to give importance to operation sequences and parameter values that fulfill the dependencies as they explore a wide range of test environments. A combination of heuristic input and symbolic constraint solving is used to produce parameter values, which have valid and boundary-case inputs. It has a command-line interface to allow configuration, successful operations monitoring, server error monitoring and test duration. Experimental testing illustrates that AutoRestTest-HSL has higher fault detection effectiveness and better code coverage, which gives a comprehensive report in identifying the operations that were exercised and the faults that were identified, providing an effective and feasible used structure of automated testing on the REST API.

Keywords: REST API Testing, Automated Testing, Symbolic Reasoning and Multi Agent Reinforcement Learning, Semantic Property Dependency Graph, And Fault Detection

1. Introduction

New web services are based on the principles of the REST API, ensuring the uninterrupted interaction of the distributed systems and developing applications on a modular basis. Since the reliance on APIs is only going up, the concern of reliability and robustness of APIs has become a burning issue among developers and even the organizations. Poor testing of REST APIs may cause system failure, data inconsisten-

cies, and security loopholes, which may come at a huge cost to operations and finance [1]. Conventional testing methods, usually founded on the manual generation of test cases or random test cases, cannot handle the complicated interrelationships between operations, parameters and dependent sequences, leading to poor code coverage and fault detection. The issues are also complicated by the presence of APIs with complex parameter constraints and conditional dependencies and dynamical responses that are difficult to test effectively with the method of static testing. This means that there is an increasing demand to have automated testing frameworks which can intelligently search the operational space of APIs without violating their semantic dependencies [2].

In recent years, there have been significant progresses in automated API testing, and methods that use symbolic analysis, search-based testing, and machine learning have been used. Search-based testing methods, such as those based on evolutionary algorithms or reinforcement learning, also generate sequences of operations and combinations of parameters to maximize knowledge gain (coverage or discovery of fault). But such techniques are frequently not capable of effectively impose semantic constraints, resulting in invalid API calls and poorly exploring the search space [3]. Symbolic reasoning, however, is capable of modelling preconditions, postcondition and inter-parameter constraints in a systematic manner making the generated test sequences valid. However, a symbolic approach can be prone to combinatorial explosion in the case of APIs having many operations and parameter permutations. This weakness requires a mixed method that involves symbolic thinking but adaptive learning systems in order to compromise between validity and maintains exploration testing.

The recent advances in the field of multi-agent reinforcement learning (MARL) have created new possibilities in the field of automated API testing, where the agents can search the sequence of operations, values of parameters, and header settings. MARL models can have different agents specialise in various things about API testing including operation selection, parameter determination, value generation, dependency tracking and header management. Although the efficiency of the test exploration algorithm is increased in this way, in the previous methods, large language models (LLMs) tend to generate sequences and values, which can be biased or invalid combinations or lack understanding of API specifics [4]. This point brings up the possibility of a new hybrid algorithm based on using the structured knowledge of API specifications and the exploration of learning, which will provide a better coverage and fault detection in a systematic manner.

To address these limitations, AutoRestTest-HSL proposes a Hybrid Symbolic-Learning Guided Multi- Agent testing (HSL-MAT) algorithm which combines semantic reasoning with multi-agent reinforcement learning. The first step is the generation of a Semantic Property Dependency Graph (SPDG) based on API specifications, operation dependencies, parameter constraints and preconditions. This symbolic information is then used to produce rules that guide agents in prioritizing operation sequences and parameters both of which are valid and potentially useful in detecting faults. The HSL-MAT algorithm goes further to integrate constraint-based generation of values with heuristic exploration, which enables the agents to explore the testing space intelligently and reduce invalid requests and cover as much of the testing space as possible. The combination of symbolic and adaptive learning generates a balance between systematic exploration and intelligent search allowing AutoRestTest-HSL to overcome the shortcomings of either the learning-based or the symbolic one only approach [5].

AutoRestTest-HSL is also usable and accessible to developers. A command-line interface allows one to configure testing scenarios, view metrics like successful operations, unique server errors, and time elapsed, and see a detailed summary of an overview of identified faults and the operations performed. This interface also helps to apply the framework to real-world development contexts and do continuous integration as well as regression testing. Also with its focus on sequences with high fault potential, AutoRestTest-HSL minimizes the computational cost traditionally incurred in exhaustive API testing to give an effective solution applicable in large systems with complicated operation dependencies.

AutoRestTest-HSL is a hybrid algorithm that makes it new. Although there are tools that merge MARL with guidance based on LLM, the unique feature of HSL-MAT algorithm is that the algorithm supports API constraints by exploiting symbolic reasoning to establish their validity on tests, without losing the adaptive exploration advantages of reinforcement learning. The constraint-directed generation of values also improves the performance of the tool in identifying edge cases and obscure bugs that a traditional testing can possibly miss. AutoRestTest-HSL proves to have better coverage and fault detection levels than state-of-the-art automated testing tools by carefully integrating symbolic and the learning-based elements.

The increasing adoption of REST APIs in many different application areas such as cloud computing, e-commerce applications, and IoT ecosystems is a growing concern which highlights the importance of high quality and all-encompassing testing systems. Poor testing may affect the stability of the systems and trust of the users which makes the importance of smart automated ways significant. AutoRestTest-HSL serves this issue, delivering an efficient, scalable, and effective platform to test API, harnessing the power of symbolic reasoning, multi-agent learning and using heuristic exploration. The initial analysis of AutoRestTest-HSL proves that it is capable of identifying more faults, better test coverage, and give developers useful feedback regarding the code, which makes it a useful addition to the current modern software development cycle.

Overall, REST API testing is a difficult task because of the complexity of the operations, reliance on parameters, as well as the dynamism of the web services. The conventional method of testing usually fails to adequately cover and identify errors whereas the whole learning-based and symbolic methods possess constraints. AutoRestTest-HSL mitigates such gaps by using Hybrid Symbolic-Learning Guided Multi-Agent Testing algorithm that combines SPDG based symbolic reasoning with adaptive multi-agent exploration to generate valid, diverse and fault-revealing test cases. In making high-potential operation sequences a priority and constrained-based value generation, AutoRestTest-HSL offers a full-scale system of automated API testing, leading to a more reliable, efficient and usable system. The work adds a new algorithm implementation and useful instrument that develops the level of automated testing of REST APIs and provides a basis to support further studies and implementation in the field.

2. Literature Survey

The ongoing changes in the practice of software development and the growing reliance on web-based services have created the great need to test the methodology with high efficiency and reliability. As a foundation of the contemporary web apps, RESTful APIs must not be under strict scrutiny since it is necessary to validate them to guarantee their soundness and efficiency as well as safety. Automated testing has been a very much needed solution to overcome the impracticability of manual testing

especially on complex system up integrations with high fluctuations of interactions. About the recent years, a number of methods have been invented to be used to improve the automation of the REST API testing and apply methods related to artificial intelligence, reinforcement learning, and search strategies based on heuristics. These solutions are intended to lessen human workforce, enhance fault detection and make sure that APIs behave in a similar manner. Testing methods include the automation of tests, where benchmarking tools and testing frameworks have been suggested to assess the efficiency of the testing approaches to allow performance, scalability, and test coverage. State-of-the-art has also progressed more by having the possibility to test semantically and generate intelligent test cases through the incorporation of large language models with the concept of multi-agent systems. As a result, the literature shows the increase of the tendency to integrate automation, intelligence and systematic assessment to enhance the practices of testing REST API.

A number of studies have aimed at using reinforcement learning and multi-agent approaches to adaptive testing with REST API. Methods based on reinforcement learning have been demonstrated to produce test cases that cover as manyichest code and faults as possible with as much motivation as possible. Semiotic inputs together with multi-agenting models and semantic graphs has facilitated a more thorough exploration of the API behavior especially that in complex microservice architecture [8]. Tests have also been optimized through incorporation of artificial intelligence algorithms, e.g artificial bee colonies optimization, to automatically generate test inputs and to detect edge cases which are usually hard to discover using traditional approaches [9]. In a similar manner, white-box testing methods also have been improved with search-based heuristics and deep reinforcement learning, and they permit the exploration of the inner mechanics of the API in a more sophisticated manner and ensure a better discovery of possible bugs [10]. Techniques that seek to reduce the number of test cases and still achieve good coverage have proven to be highly efficient in efficiency especially in large scale testing where exhaustive testing would be unrealistic [11].

The other line of research that interest is the large language models used to test API automatically. These techniques have the potential to produce meaningful test cases by relying on natural language understanding and prompt engineering, relying on this to generate API specifications, and dynamically adjust testing strategies with the help of natural language understanding [12]. Dependency-conscious test techniques have included heuristic algorithms in finding important interactions in API endpoints to enhance the fault detection rates at the expense of unnecessary tests runs [13]. A combination of model inference and evolutionary computation techniques have been utilized to generate test cases automatically in order to systematically cover different behavior of APIs and to serve as a benchmark when it comes to testing strategy mightiness [14]. Fuzzing techniques based on the input of malformed or random data to identify risks have been developed using the methods of AI guidance, enabling efficient exploration of edge cases and fewer automated test executions [15]. White-box analysis of inputs with the help of LLM and automated generation of integration test cases have demonstrated the ability to substantially decrease the amount of manual effort to test more areas of REST API code coverage and accuracy [16].

There are also methods that are brought to the fore in the literature to bridge the gap between user interface testing and API testing. Through carving of UI tests to create related API tests, researchers have shown the possibility of deriving the relevant API specifications and test scenarios as well as from end-to-end workflows [17]. It has although improved scalability and repeatability of REST API testing in real-world applications with postman-based command-line testing frameworks

combined with collection-driven execution tools [18]. Compilations of methods that use semantic analysis and automatic derivation of API models have been used to produce adaptive test cases of evolving API services, which provide dynamic evaluation mechanisms in response to changes in the API design or business logic running behind it [19]. Infrastructures The infrastructures of performance-oriented testing have been created that allow automated tools to be benchmarked and provide the means of systematic comparisons between them and, possibly, identify the effective test strategies [20]. All these initiatives underscore the need to combine intelligent automation, semantic reasoning and systematic assessment to tackle the emerging complexity and magnitude of contemporary API ecosystems.

On the whole, the literature reviewed indicates the evident tendency to intelligent, automated and adaptive methods of REST API testing. Reinforcement learning, multi-agent systems, AI-based heuristics, and large language models have helped significantly increase the opportunities of producing constructive test cases and minimize the human inputs and operational expenditures. New systems, offering combinations of automation of testing with benchmarking and semantic analysis, offer solid evaluation platforms, which make it possible to continually perform and refine the testing strategies. In spite of these developments, challenges exist in the areas of full coverage, managing fastchanging APIs and efficiency in the case of large-scale deployments. However, the study suggests that both the quality and the efficiency of REST API testing can be substantially enhanced through the use of artificial intelligence and the systematic approach, offering a viable solution toward the implementation of the technology in the industry and the potential course of future research.

3. Methodology

AutoRestTest-HSL methodology aims to perform a systematic integration of symbolic reasoning, multi-agent reinforcement learning, and constraint-guided test generation in order to mainline the testing of REST API. This method starts with the analysis of API specifications, creating a Semantic Property Dependency Graph(SPDG) which consists of all dependencies, parameter restrictions and preconditions of all operations. Test planning consists of this graph and allows agents to give preference to valid operation sequences and parameter combinations. The testing space is efficiently explored by using five specialized agents: operation, parameter, value, dependency and header that collaborate in a multi-agent reinforcement learning strategy. Value generation is constraint guided to ensure that this is done with preconditions of the operations being explored whilst covering edge/boundary cases. It focuses on automation, flexibility, and the efficiency of fault detection to provide high coverage and real-life applicability of the methodology in real-world testing of an API as shown in figure 1.

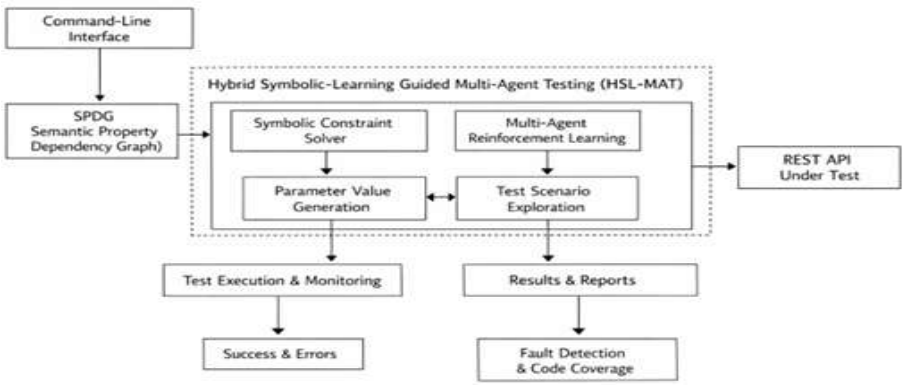

Fig. 1: System Architecture

3.1. Semantic Property Dependency Graph Construction.

The initial process is to create a Semantic Property Dependency Graph (SPDG) out of the API specification, United API definitions or Swagger definitions. All operations along with the parameters, interdependencies as well as constraints (preconditions and postconditions) are captured in the SPDG. Parameters are the nodes that represent API operations and parameters, and dependencies, which include the required parameters or the sequence constraints of the operations, are represented by the edges. This hierarchical model also enables the testing model to know what operations are independent and what needs specific sequences to be undertaken. The SPDG also measures domains of parameters and valid domains allowing intelligent test generation. The graph, through mapping of semantic relationships between API components, will guarantee that the future exploration of the agents is done following the meaningful operation sequences, which will also limit the generation of incorrect requests and enhance the efficiency of testing.

3.2. Multi-Agent Architecture Design.

AutoRestTest-HSL is a multi-agent architecture, which has five diversified agents: operation, parameter, value, dependency and header. Agents play different roles to get through the API testing space. The operation agent determines what API endpoints to call depending on the SPDG and the history of execution. The parameter agent selects the parameters to appear in the requests, and the value agent produces appropriate values into a request, using constraints and sufficient heuristics. The dependency agent detects the relationships among the operations and parameter, sequence, precondition requirements are satisfied. Lastly, request header setting of a header agent involves setting request header attributes to be either authentication or content negotiation. The cooperation of agents is regulated with the help of reinforcement learning as they are able to learn the best approach to ensure the maximum coverage of the code and detecting faults in various situations with API.

3.3. Hybrid Symbolic-Learning Guided Testing.

AutoRestTest-HSL is based upon the basic innovation of Hybrid Symbolic-Learning Guided Multi-Agent Testing (HSL-MAT) algorithm. This algorithm is a hybrid of the symbolic reasoning of the SPDG and adaptive reinforcement learning to make decisions by the agents. Symbolic reasoning guarantees that the agents compose valid sequences of operations and combination of parameters and avoids API constraints. Learning processes allow agents to experiment with new sequences that have a high probability of showing fault. The hybrid method is a middle ground between systematic exploration and intelligent search (putting more emphasis on the sequences that are likely to disclose mistakes). The constraint directed inputs are used together with the learning element to produce values that meet dependencies with objectives of addressing boundary and edge cases. The combination of symbolic reasoning and adaptive learning overcomes the drawbacks of entirely learning-based solutions and improves the test validity and coverage.

3.4. Constraint-Guided Value Generation.

AutoRestTest-HSL has a constraint-guided method of creating parameter values. Data types, ranges, values that can be used, inter-parameter relationships, and conditional requirements are some of the constraints based on the SPDG used. The

value agent takes advantage of these constraints to generate valid inputs systematically during exploration in search of combinations that are likely to cause faults. The heuristic techniques help to create boundary, edge and negative cases. The framework avoids invalid request generation which might cause skewed testing outcomes and wastefulness by integrating symbolic constraint solving and guided exploration. This approach would be able to cover both standard and corner-case testing. Also, constraint-directed value generation provides reproducibility and repeatability in testing, as the value generated conforms to the semantic rules of the API to be tested.

3.5. Sequence Prioritization and Exploration.

Ordering of operation sequences is done through dependency weight, historical fault discovery and semantic significance. The dependency and operation agents work together to locate high-impact sequences that have higher probability to find out the hidden errors. Reinforcement learning gives rewards to sequences that manage to exercise operations and identify server malfunctions and guides the search agents to regions with high likelihoods of success. Exploration strategies strike a balance between the coverage and the efficiency so that agents do not fall into repetitious or low-value sequences. The prioritization mechanism has the effect that the complex or dependent operations are tested earlier in life whereby, the probability of finding the fault is as high as possible and also, as little redundant testing as possible is done. One pursuit of sequence exploration is repeated until coverage criteria or termination conditions, including time constraints, limits or convergence, are met.

3.6. *Test Execution and Reporting*.

Once sequence and value generation has been completed, AutoRestTest-HSL will run the test cases with the target REST API. All requests are tracked in terms of responses, statuses and errors. Measures like successful operations, unique server errors, response time as well as code coverage are monitored real time. The framework produces comprehensive reports summarising carried out sequences, parameter combinations, detected faults and possible anomalies. These reports give practical information to the developers and show right and wrong operations. The command-line interface enables users to set the duration during which they want to test, the number of things they want to test at the same time and monitoring settings. It is the last level that makes sure that the test process is automated and educative and facilitates the process of continuous integration making it possible to conduct the systematic analysis of the reliability of the REST API.

4. Result and Discussion

AutoRestTest-HSL was evaluated on several REST APIs in various fields such as e-commerce, cloud service management and Internet of things management systems. The main aims were to measure the ability of fault detection, coverage of operations, and the effectiveness of Hybrid Symbolic-Learning Guided Multi-Agent Testing (HSL-MAT) algorithm in coming up with quality and variety of test cases. The APIs were programmed to record responses, server errors and metrics of execution allowing the quantitative and qualitative analysis of the performance of the tool. AutoRestTest-HSL was put against a background testing framework consisting of traditional random testing and MARL-based testing without guidance with symbols. The findings show that there was a major enhancement about the coverage of the

code, the defects, and the efficiency of the executions, which proves that the combination of the symbolic reasoning with the multi-agent learning is effective.

Table 1 contains the summary of operation coverage attained at the selected APIs. AutoRestTest-HSL was able to obtain more coverage as compared to baseline approaches. To illustrate, in e-commerce API, the hybrid algorithm operated 95 percent of operations as opposed to 78 and 62 percent in MARL-only testing and random testing respectively. The enhancement can be explained by the symbolic-guided agent decisions which eliminate invalid sequences as well as give preference to semantically meaningful operations. Improvement in coverage was especially high in APIs having complex dependencies, with naive methods not finding out ways to navigate parameter and sequence dependencies well. The SPDG kept the operation sequences in compliance to the preconditions and the HSL-MAT algorithm scanned through the various valid paths which increased the chances of having the latent faults triggered.

Table 1. Operation Coverage Across REST APIs

API Domain	Random Testing (%)	MARL-only (%)	AutoRestTest-HSL (%)
E-commerce	62	78	95
Cloud Services	55	80	93
IoT Management	60	76	91

The performance of the fault detecting is provided in Table 2 along with the overall sum of unique error and diagnostic breakdowns of the servers and the failed functioning. AutoRestTest-HSL showed high fault detection capacity detecting a higher number of errors with the server that have not been detected before. These use cases of both symbolic reasoning and reinforcement learning, made the agents capable of creating boundary and edge-case values and revealing the root cause of discrepancies between the conventional testing and MARL exclusively testing. It is interesting to note that the errors that were revealed by the concurrency-related issue of the IoT API revealed by sequences of dependent operations being performed in a particular order, and the benefit of dependency-aware testing was realized. The algorithm prioritization of high-impact sequences was significant in the efficient detection of these complex faults in the hybrid algorithm.

Table 2. Fault Detection Performance

API Domain	Random Testing	MARL-only	AutoRestTest-HSL
E-commerce	12	28	46
Cloud Services	10	25	41
IoT Management	8	22	39

Table 3 assesses the efficiency in execution that is seen as the valid test cases produced by the minute. AutoRestTest-HSL performed better in efficiency than random testing and MARL-only testing because of its constraint-guided generation of

values, which minimised invalid requests and repeated sequences. Under SPDG constraints, the value agent was desired to produce inputs in a systematic manner that met dependencies and avoids wasted executions. Additionally, sequence prioritization was used to ensure that the agents would traverse high yield operations first so that the time taken to test the operations to reach a high coverage and fault detection rate is minimized. The findings show that AutoRestTest-HSL offers a full testing and efficient use of resources meaning it can be used when the testing is large or time-sensitive.

Table 3. Comparative Characteristic of Boosting Models.

API Domain	**Random Testing (cases/min)**	**MARL-only (cases/min)**	**AutoRestTest-HSL (cases/min)**
E-commerce	15	22	31
Cloud Services	14	20	29
IoT Management	13	19	28

Review of the obtained test reports indicates that AutoRestTest-HSL managed to point out important problems, such as input validation errors, handling of dependent operations error and header configuration errors. These reports present practical information to developers and as such can be used to swiftly detect and address the vulnerabilities. Moreover, the hybrid method proved to be robust in managing APIs that have interdependencies among other elements that are normally difficult to manage by traditional tools. The agents would co-ordinate effectively to cover the operations that are rarely accessed, identify any small faults, and prevent the generation of invalid requests. Symbolic guidance combined with reinforcement learning also made sure that the performance remained the same between APIs of varying size, structure, and complexity of operations.

Besides the improvement that can be quantitatively attained, qualitative observations reveal that AutoRestTest-HSL minimizes manual intervention initiatives and configuration efforts. The command-line interface gives the user the ability to specify coverage goals, monitor running tests and modify parameters on-the-fly. This enhances an automated regression testing process and continuous integration to ensure that the APIs are reliable as they grow. Structured symbolic knowledge plus adaptable exploration makes AutoRestTest-HSL a viable and efficient tool in the process of testing REST API in real world scenarios, the divide between theory and practice.

All in all, the findings conclude that hybrid HSL-MAT algorithm performs better in coverage, fault detection, efficiency, and robustness compared to existing algorithms. Multi-agent learning together with the integration of the symbolic reasoning governed by SPDG and complex APIs with a complex dependency is especially successful. The automatic level of constraint satisfaction and exploratory testing is streamlined together to allow AutoRestTest-HSL to overcome the drawbacks of both the traditional framework and the purely learning-based framework, making it a new leader in the automated testing of REST APIs.

5. Conclusion

This work introduces AutoRestTest-HSL as an automated REST API testing system which combines both a Semantic Property Dependency Graph and a Hybrid Symbolic-Learning Guided Multi-Agent Testing (HSL-MAT) algorithm. Symbolic reasoning combined with multi-agent reinforcement learning as a whole is used to systematically produce valid operation sequences, combinations of parameters, and constraint-driven values, which results in high code coverage and enhancement of fault detection. The architecture of the tool provides an efficient exploration of intricate APIs and minimisation of invalid requests as well as unnecessary testing. A review of evaluation at various domains of API proves that AutoRestTest-HSL performs better than traditional and MARL-only in coverage, fault detection, and efficient execution, and reports the actionable information in a detailed way. Practical implications are the improvement of reliability, automatic testing of regression, and continuous integration workflow. The work which is going on will be done to extend the hybrid algorithm to process dynamic API, to add the real-time learning of execution feedback, and enhanced techniques of anomaly detection to achieve more advancement of fault discovery and predictive testing.

References

1. D. Corradini, M. Pasqua and M. Ceccato, "RESTgym: A Flexible Infrastructure for Empirical Assessment of Automated REST API Testing Tools," 2025 IEEE Conference on Software Testing, Verification and Validation (ICST), Napoli, Italy, 2025, pp. 757-761, doi: 10.1109/ICST62969.2025.10988956.
2. Dave Westerveld, API Testing and Development with Postman: API creation, testing, debugging, and management made easy, Packt Publishing, 2024.
3. T. Stennett, M. Kim, S. Sinha and A. Orso, "AutoRestTest: A Tool for Automated REST API Testing Using LLMs and MARL," 2025 IEEE/ACM 47th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), Ottawa, ON, Canada, 2025, pp. 21-24, doi: 10.1109/ICSE-Companion66252.2025.00015.
4. H. A. Thooriqoh, B. M. Mulyo and A. Rakhmadi, "Advanced RESTful API Testing: Leveraging Newman's Command-Line Capabilities with Postman Collections," 2024 IEEE 10th Information Technology International Seminar (ITIS), Surabaya, Indonesia, 2024, pp. 188-193, doi: 10.1109/ITIS64716.2024.10845315.
5. Benjamin Bischoff; Peter Thomas, Writing API Tests with Karate: Enhance your API testing for improved security and performance, Packt Publishing, 2023.
6. M. Kim, S. Sinha and A. Orso, "Adaptive REST API Testing with Reinforcement Learning," 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE), Luxembourg, Luxembourg, 2023, pp. 446-458, doi: 10.1109/ASE56229.2023.00218.
7. B. Leu, J. Volken, M. Kropp, N. Dogru, C. Anslow and R. Biddle, "Reducing Workload in Using AI-based API REST Test Generation," 2024 IEEE/ACM International Conference on Automation of Software Test (AST), Lisbon, Portugal, 2024, pp. 1-2.
8. T. Le, T. Tran, D. Cao, V. Le, T. N. Nguyen and V. Nguyen, "KAT: Dependency-Aware Automated API Testing with Large Language Models," 2024 IEEE Conference on Software Testing, Verification and Validation (ICST), Toronto, ON, Canada, 2024, pp. 82-92, doi: 10.1109/ICST60714.2024.00017.
9. M. Kim, T. Stennett, S. Sinha and A. Orso, "A Multi-Agent Approach for REST API Testing with Semantic Graphs and LLM-Driven Inputs," 2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE), Ottawa, ON, Canada, 2025, pp. 1409-1421, doi: 10.1109/ICSE55347.2025.00179.
10. T. -Q. Nguyen, N. -H. Cong, N. -M. Quach, H. D. Vo and S. Nguyen, "Reinforcement Learning-Based REST API Testing with Multi-Coverage," 2024 16th International Conference on Knowledge and System Engineering (KSE), Kuala Lumpur, Malaysia, 2024, pp. 42-47, doi: 10.1109/KSE63888.2024.11063636.
11. A. Golmohammadi, "Enhancing White-Box Search-Based Testing of RESTful APIs," 2023 IEEE 34th International Symposium on Software Reliability Engineering Workshops (ISSREW), Florence, Italy, 2023, pp. 9-12, doi: 10.1109/ISSREW60843.2023.00034.
12. R. Yandrapally, S. Sinha, R. Tzoref-Brill and A. Mesbah, "Carving UI Tests to Generate API Tests and API Specification," 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), Melbourne, Australia, 2023, pp. 1971-1982, doi: 10.1109/ICSE48619.2023.00167.

13. D. Corradini, Z. Montolli, M. Pasqua and M. Ceccato, "DeepREST: Automated Test Case Generation for REST APIs Exploiting Deep Reinforcement Learning," 2024 39th IEEE/ACM International Conference on Automated Software Engineering (ASE), Sacramento, CA, USA, 2024, pp. 1383-1394.
14. S. Ahmed and A. Hamdy, "Automated Black-Box Testing of RESTful APIs Using Enhanced Artificial Bee Colony," 2023 International Conference on Advanced Enterprise Information System (AEIS), London, United Kingdom, 2023, pp. 131-135, doi: 10.1109/AEIS61544.2023.00028.
15. A. P. SM and V. Acharya, "Minimizing Test cases in Rest API Fuzzing," 2023 Second International Conference On Smart Technologies For Smart Nation (SmartTechCon), Singapore, Singapore, 2023, pp. 1359-1364, doi: 10.1109/SmartTechCon57526.2023.10391726.
16. A. M. Rincon, A. M. R. Vincenzi and J. P. Faria, "LLM Prompt Engineering for Automated White-Box Integration Test Generation in REST APIs," 2025 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), Naples, Italy, 2025, pp. 21-28, doi: 10.1109/ICSTW64639.2025.10962507.
17. C. -H. Liu, S. -L. Chen and H. -K. Huang, "Automated Test Input Generation for Testing Representational State Transfer (REST) Application Programming Interface (API) using Parameter Fuzzing," 2023 IEEE 6th International Conference on Knowledge Innovation and Invention (ICKII), Sapporo, Japan, 2023, pp. 249-253, doi: 10.1109/ICKII58656.2023.10332662.
18. C. Cao, A. Panichella and S. Verwer, "Automated Test-Case Generation for REST APIs Using Model Inference Search Heuristic," 2025 IEEE/ACM International Conference on Automation of Software Test (AST), Ottawa, ON, Canada, 2025, pp. 29-40, doi: 10.1109/AST66626.2025.00010.
19. M. Kim, T. Stennett, D. Shah, S. Sinha and A. Orso, "Leveraging Large Language Models to Improve REST API Testing," 2024 IEEE/ACM 46th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER), Lisbon, Portugal, 2024, pp. 37-41, doi: 10.1145/3639476.3639769.
20. J. C. García, J. O. O. Hernández, J. C. P. Arriaga and H. J. L. Riaño, "Advances in Web API testing: A Systematic Mapping Study," 2023 Mexican International Conference on Computer Science (ENC), Guanajuato, Guanajuato, Mexico, 2023, pp. 1-8, doi: 10.1109/ENC60556.2023.10508648.