RESEARCH ARTICLE OPEN ACCESS

API TESTING

Krati Joshi*

*(B-tech (CSE) Parul University,Vadodara

210303105498@paruluniversity.ac.in)

--************************----------------------------------

Abstract:

This APIs are fundamental to modern software systems, facilitating seamless communication between applications. Ensuring their reliability and correctness requires rigorous testing. This paper presents an automated API testing framework that leverages OpenAPI schemas and AI-driven test case generation using OpenAI's GPT-4o model. The framework systematically tests REST API methods, including GET, POST, PUT, PATCH, and DELETE, by dynamically generating and executing test cases. It intelligently handles incomplete API documentation, reference parameters, and error response analysis, ensuring comprehensive validation. Our approach automates test execution, captures real-time logs, and refines test cases iteratively based on API responses. The system efficiently detects and reports errors, including **4XX** and **5XX** status codes, while updating partially documented schemas. Despite challenges such as handling dynamic parameters, AI-generated test case accuracy, and scalability concerns, the framework significantly enhances API testing efficiency. Future improvements aim to refine test case generation, optimize performance, and support broader API testing scenarios.

Keywords — **API Testing, OpenAPI, AI-driven Testing, RESTful APIs, GPT-4o, Automated Testing, API Validation**

--************************----------------------------------

I. INTRODUCTION

In modern software development, API testing plays a crucial role in ensuring seamless communication between different systems and services. With the growing complexity of web applications, manually testing APIs becomes inefficient, time-consuming, and prone to human error. To address these challenges, API Testing Automation provides a scalable and reliable approach to validating API functionality, security, and performance. This internship focuses on automating API testing using the OpenAPI schema (JSON) to streamline the process of test case generation and execution.

The project leverages advanced technologies, including AI-driven automation with OpenAI's GPT-4o, to dynamically generate and validate test cases for REST APIs. It covers essential aspects such as GET, POST, PUT, PATCH, and DELETE requests, response validation, and error handling.

The framework automates API calls, analyzes responses, and logs test results in CSV format, allowing for structured data representation and easy debugging. This approach not only enhances the efficiency of testing but also ensures highquality API development by identifying potential issues early in the software lifecycle.

Through this project, I aim to gain hands-on experience in automated testing, AI integration, and API validation, while contributing to the development of a robust, scalable, and intelligent API testing framework.

II. BENEFITS AND IMPACT

Improved Efficiency: Automation reduces manual testing efforts, increasing speed and accuracy.

 ©IJSRED: All Rights are Reserved

• Enhanced API Reliability : AI-driven test generation ensures thorough validation and robust API performance.
• Scalability: The framework can be expanded to support large-scale API testing across multiple endpoints.
• Error Handling & Documentation: Systematic logging of test cases, responses, and errors aids in debugging and optimization.
• Future Adaptability: The approach supports AI-driven improvements, making the framework adaptable to evolving software testing needs.

III. LITERATURE REVIEW

API testing plays an important role in ensuring the reliability of web services. As APIs become more complex, traditional manual testing methods struggle to keep up, leading to the need for automated testing approaches. Researchers have explored different ways to make API testing more effective and efficient.

One of the most well-known tools for automated API testing is EvoMaster, introduced by Arcuri [5], [4]. It uses search-based techniques to generate test cases and find potential issues in REST APIs. Similarly, Cao et al. [1] proposed an approach that uses model inference and search heuristics to improve test case generation, helping detect API failures more effectively.

Artificial Intelligence (AI) and machine learning have also been used to improve API testing. Pereira et al. [2] developed APITestGenie, a tool that uses generative AI to create API test cases automatically. Their work shows how AI can make testing faster and more reliable. Kim et al. [3] also studied the challenges of current REST API testing methods and suggested improvements to make automated test generation more effective.

Another approach to API testing involves learning API behavior from data. Baumgartner and Verwer [8] introduced a method that learns state machines from API interactions, allowing for better test case generation. Test case selection and prioritization are also important for making testing more efficient. Yoo and Harman [9], [10] worked on techniques to reduce the number of test cases while ensuring important tests are not missed.

Evolutionary algorithms have also been explored for test generation. Črepinšek et al. [11] reviewed different strategies to balance exploration and exploitation in software testing, which helps improve the quality of generated test cases. Additionally, Hajri et al. [12] introduced a method to classify and prioritize test cases in API-based systems, making the testing process smoother.

To make API testing more structured, many tools now use the OpenAPI Specification, which provides a standard way to define APIs. This helps automate the generation of test cases and improves compatibility between different testing tools [7].

In summary, researchers have made significant progress in API testing using automated tools, AI, machine learning, and evolutionary algorithms. However, there are still challenges, such as handling dynamic APIs, improving test selection methods, and ensuring test diversity. Future research will continue to focus on these areas to make API testing even more effective.

IV. METHODOLOGY

In this section, we describe the approach used to automate API testing based on the OpenAPI schema. The methodology consists of multiple phases, including test case generation, API execution, response validation, and iterative improvement using AI-based assistance.

A. Input Data

The primary input to our system is the OpenAPI schema, which is provided in JSON format. The schema may be either fully completed or partially completed. It defines the structure of the APIs,

 ©IJSRED: All Rights are Reserved

including endpoints, parameters, request methods, response formats, and constraints.

B. Testing Approach

The testing process begins with the validation of GET requests, as these are independent and do not rely on other APIs. The methodology follows a structured workflow as described below:

1. **Schema Preparation:** We create sample OpenAPI schemas that include various types of GET requests. These schemas contain:

    - Parameters of different data types, such as lists, integers, floats, and strings.
    - Dynamic path parameters (e.g., /path/{custom id}/) to simulate real-world API usage.
    - Request bodies, even though they are not typically used in GET requests.

2. **Test Case Generation:** Using OpenAI's GPT-4o model, we generate an initial test case based on the available schema details. The test case includes all necessary request parameters and valid input values.
3. **API Execution and Response Validation:** The generated test case is executed by making an API call and analyzing the server response:

    - If the response status code is **2XX**, the test is considered successful, and no further modifications are needed.
    - If the response status code is **4XX** (client error), OpenAI is used again to generate a modified test case by incorporating the previous test case and error response as context.
    - If the response status code is **5XX** (server error), the testing process is halted, and the error is reported to the user for manual intervention.

4. **Logging API Calls:** Each API call is logged into a CSV file with the following fields:

    - API URL (path)
    - Test Case (JSON format)
    - Status Code
    - Response (JSON or text in case of decoding errors)
    - Action taken (Retry, Stop, or OpenAI-based correction)

5. **Iterative Test Case Enhancement:** Once a successful response (**2XX**) is received, the OpenAPI schema is updated if it was incomplete. Additional test cases are then generated programmatically by:

    - Creating logical variations based on parameter constraints defined in the schema.
    - Testing boundary values, such as negative numbers, zero, and extreme values for integer parameters.
    - Passing out-of-range or invalid values to evaluate API robustness.

6. **Handling Reference Parameters:** One of the major challenges encountered during testing was managing reference parameters within test cases. This issue was addressed by refining the test case generation logic and leveraging AI-based techniques for adaptive test modifications.

C. Technologies and Tools Used

The methodology is implemented using Python as the primary programming language. The key technologies and tools used include:

- **OpenAPI Schema:** Used for defining API specifications and generating test cases.
- **GPT-4o Model:** Utilized for intelligent test case generation and iterative improvements.
- **CSV Logging:** Ensures structured recording of API test results.

- **Python Libraries:** Includes requests for API execution, JSON parsing libraries, and automation tools for processing API responses.

D. System Architecture

The system architecture follows a structured workflow for automated API testing. It consists of multiple components, including OpenAPI schema processing, test case generation using AI, API execution, response validation, and iterative improvements. The architecture ensures efficient logging and error-handling mechanisms to refine test cases dynamically.

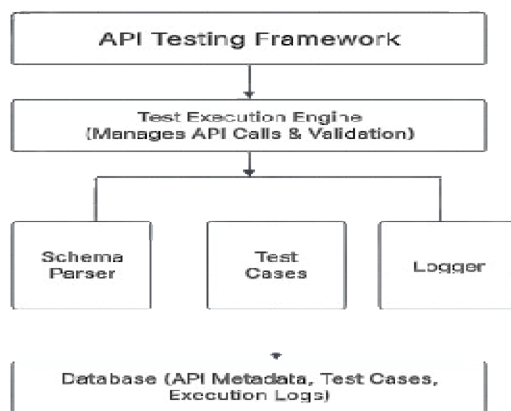

Fig 1. System Architecture

V. RESULTS AND DISCUSSION

The implementation of the automated API testing framework yielded promising results, demonstrating its capability to systematically test API endpoints based on the OpenAPI schema. The testing process involved executing various API methods, analyzing responses, and iteratively refining test cases based on errors encountered.

A. Execution Summary

During testing, multiple API endpoints, including those handling user management, store operations, and pet-related services, were processed. The framework successfully executed a series of REST API methods such as **GET, POST, PUT,** and **DELETE**.

A majority of the test cases executed returned successful responses with **2XX** status codes, indicating correct functionality. However, some test cases encountered errors in the form of **4XX** and **5XX** status codes. These errors were primarily due to missing or incorrect input parameters, authorization issues, or server-side failures.

The framework dynamically generated and modified test cases based on response feedback. If a request failed with a **4XX** status, the system refined the test case using OpenAI's GPT-4o model, adjusting input parameters accordingly. In contrast, if a **5XX** status was encountered, the system flagged the issue for manual review, as server-side errors were beyond the scope of automated resolution.

B. Analysis of Challenges

While the framework performed effectively, several challenges were identified:

- **Handling Reference Parameters:** Certain APIs required dynamic values such as user IDs or order numbers, which were dependent on prior API calls. Ensuring proper sequencing and maintaining consistency across test cases proved to be a challenge.
- **Incomplete OpenAPI Schema:** Some endpoints lacked sufficient documentation, making it difficult to infer required parameters and expected responses, leading to failed test executions.
- **Error Variability:** API responses contained diverse error messages, requiring advanced handling to accurately interpret and modify test cases.
- **Response Latency:** The process of dynamically generating test cases using OpenAI introduced minor delays, especially

 ©IJSRED: All Rights are Reserved

when multiple iterations were needed for test refinement.

C. Potential Enhancements

To improve the framework's efficiency and accuracy, the following enhancements are proposed:

- **Reference Parameter Management:** Implementing a caching system to store and reuse dynamically generated values for improved test consistency.
- **Schema Validation Mechanism:** Enhancing pre-execution validation to detect missing schema details and auto-fill default values where possible.
- **Optimized Error Handling:** Introducing rule-based adjustments alongside AI-generated modifications to handle a wider range of failure cases.
- **Reducing Dependency on OpenAI:** Incorporating heuristic-based test generation methods to minimize reliance on external AI models, thereby improving execution speed.

VI. CHALLENGES AND LIMITATIONS

During the implementation of the automated API testing framework, several challenges were encountered, which affected efficiency and accuracy. These limitations highlight areas for future improvements.

A. Handling Reference Parameters

One major challenge was dealing with reference parameters in API requests, such as dynamic IDs (e.g., user IDs, order IDs). Generating valid test cases was difficult due to:

- The dynamic nature of reference values.
- Dependencies between API calls requiring prior responses.
- Frequent **4XX** errors due to missing or invalid reference values.

B. Incomplete OpenAPI Schema

The framework relied on OpenAPI schemas, but many were partially documented, leading to:

- Missing details about required and optional fields.
- Undefined constraints for input values.
- Ambiguities in expected response formats.

C. Handling 5XX Errors

The framework stopped execution upon encountering **5XX** errors. However, these errors were sometimes caused by:

- Unexpected inputs leading to server crashes.
- Discrepancies between documentation and implementation.
- Temporary server downtime affecting test reliability.

D. Limitations of AI-Generated Test Cases

The use of OpenAI's GPT-4o for generating test cases presented some challenges:

- AI-generated test cases were sometimes redundant or unrealistic.
- Complex business logic APIs were difficult for AI to handle.
- Programmatically generated test cases were often more reliable.

E. Performance and Scalability Issues

Testing large API schemas led to performance bottlenecks due to:

- High request volume within short intervals.
- Increased execution time for large schemas.
- Log file sizes growing significantly, requiring better management.

Despite these challenges, the framework effectively automated API testing and provided meaningful

 ©IJSRED: All Rights are Reserved

insights. Addressing these limitations will further enhance its accuracy, efficiency, and scalability.

VII. CONCLUSION AND FUTURE WORK

A. Conclusion

This study presented an automated API testing framework leveraging OpenAPI schemas and AI-powered test case generation to validate REST API functionality. The framework successfully processed various API methods, dynamically refined test cases based on responses, and logged execution results systematically.

The results demonstrate that automated testing significantly improves API validation efficiency by reducing manual effort and enhancing test coverage. By iteratively generating and adjusting test cases, the system effectively handled a range of expected and edge-case scenarios. However, challenges such as handling reference parameters, incomplete API documentation, and response variability highlighted areas for further optimization.

Despite these challenges, the framework provided a structured and scalable approach to API testing, ensuring robust validation of API endpoints with minimal manual intervention.

B. Future Work

Although the framework successfully implemented dynamic test case generation and execution, there are several areas where further enhancements can be made:

- **Enhanced Reference Parameter Handling:** Introducing a dependency-tracking mechanism to automatically fetch and use reference parameters from prior API responses.
- **Schema Auto-Completion:** Implementing an intelligent pre-processing module that detects incomplete OpenAPI schemas and suggests potential missing elements.
- **Performance Optimization:** Reducing the dependency on AI-based test case generation by integrating heuristic-based techniques for common test scenarios.
- **Support for Additional HTTP Methods:** Extending the framework to cover **PUT, PATCH,** and **DELETE** requests with comprehensive test scenarios.
- **Parallel Execution for Large-Scale Testing:** Implementing parallel test execution to improve efficiency when dealing with extensive API suites.
- **Integration with CI/CD Pipelines:** Seamlessly incorporating the framework into continuous integration and deployment pipelines to enable real-time API testing in software development workflows.

By addressing these areas, the framework can evolve into a more efficient and intelligent API testing solution, making it even more valuable for developers and QA teams.

REFERENCES

[1] C. Cao, A. Panichella, and S. Verwer, "Automated Test-Case Generation for REST APIs Using Model Inference Search Heuristic," arXiv preprint, 2024. [Online]. Available: https://arxiv.org/abs/2412.03420 [Accessed: Mar 4, 2025].

[2] A. Pereira, B. Lima, and J. P. Faria, "APITestGenie: Automated API Test Generation through Generative AI," arXiv preprint, 2024. [Online]. Available: https://www.arxiv.org/abs/2409.03838 [Accessed: Mar 4, 2025].

[3] M. Kim, Q. Xin, S. Sinha, and A. Orso, "Automated Test Generation for REST APIs: No Time to Rest Yet," arXiv preprint, 2022. [Online]. Available: https://arxiv.org/abs/2204.08348 [Accessed: Mar 4, 2025].

 ©IJSRED: All Rights are Reserved

[4] A. Arcuri, “RESTful API automated test case generation with EvoMaster,” ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 28, no. 1, pp. 1–37, 2019. [Online]. Available: https://arxiv.org/abs/1901.01541 [Accessed: Mar 4, 2025].

[5] A. Arcuri, “Evomaster: Evolutionary multi-context automated system test generation,” in 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST), Västerås, Sweden, pp. 394–397, 2018.

[6] A. Arcuri, “RESTful API automated test case generation,” in 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS), Prague, Czech Republic, pp. 9–20, 2017.

[7] SmartBear Software, “OpenAPI Specification.” [Online]. Available: https://swagger.io/specification/ [Accessed: Mar 4, 2025].

[8] R. Baumgartner and S. Verwer, “Learning state machines from data streams: A generic strategy and an improved heuristic,” in Proceedings of the 16th International Conference on Grammatical Inference, vol. 217, PMLR, pp. 117–141, Jul. 2023.

[9] S. Yoo and M. Harman, “Pareto efficient multi-objective test case selection,” in Proceedings of the 2007 International Symposium on Software Testing and Analysis (ISSTA ’07), ACM, New York, NY, USA, pp. 140–150, 2007.

[10] S. Yoo and M. Harman, “Regression testing minimization, selection and prioritization: a survey,” Software Testing, Verification and Reliability, vol. 22, no. 2, pp. 67–120, 2012.

[11] M. Črepinšek, S.-H. Liu, and M. Mernik, “Exploration and exploitation in evolutionary algorithms: A survey,” ACM Computing Surveys, vol. 45, no. 3, Jul. 2013. [Online]. DOI: https://doi.org/10.1145/2480741.2480752.

[12] I. Hajri, A. Goknil, F. Pastore, and L. C. Briand, “Automating System Test Case Classification and Prioritization for Use Case-Driven Testing in Product Lines,” 2020. [Online]. Available: https://arxiv.org/abs/1905.11699 [Accessed: Mar 4, 2025].

 ©IJSRED: All Rights are Reserved