ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

Công Nghĩa Hiếu

XÂY DỰNG CÔNG CỤ PHÂN TÍCH MÃ NGUỒN CHO NGÔN NGỮ RUST

KHOÁ LUẬN TỐT NGHIỆP ĐẠI HỌC

Ngành: Công nghệ thông tin

HÀ NỘI - 2024

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

Công Nghĩa Hiếu

XÂY DỰNG CÔNG CỤ PHÂN TÍCH MÃ NGUỒN CHO NGÔN NGỮ RUST

KHOÁ LUẬN TỐT NGHIỆP ĐẠI HỌC

Ngành: Công nghệ thông tin

Cán bộ hướng dẫn: TS. Võ Đình Hiếu

HÀ NỘI - 2024

VIETNAM NATIONAL UNIVERSITY, HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Cong Nghia Hieu

DEVELOPING A SOURCE CODE ANALYSIS TOOL FOR THE RUST PROGRAMMING LANGUAGE

BACHELOR'S THESIS

Major: Information technology

Supervisor: Dr. Vo Dinh Hieu

HANOI - 2024

Lời cảm ơn

Lời đầu tiên, tôi xin gửi lời cảm ơn thầy TS. Võ Đình Hiếu, người đã tận tình chỉ bảo tôi trong suốt quá trình học tập tại trường và đặc biệt là thời gian thực hiện khóa luận tốt nghiệp. Thầy đã không chỉ cung cấp những kiến thức chuyên môn quý báu mà còn động viên và hỗ trợ tôi vượt qua những khó khăn trong quá trình thực hiện khóa luận.

Tôi cũng xin cảm ơn ThS. Trần Mạnh Cường, những người đã luôn giúp đỡ tôi trong suốt thời gian thực tập. Sự hỗ trợ nhiệt tình của thầy đã giúp tôi có thêm tự tin và khả năng áp dụng kiến thức vào thực tiễn.

Ngoài ra, tôi xin gửi lời cảm ơn đến các thầy cô, anh chị, và các bạn trong phòng thí nghiệm bộ môn Công nghệ phần mềm, những người đã giúp đỡ tôi rất nhiều trong việc mở rộng kiến thức và kỹ năng thực hành.

Cuối cùng, tôi xin cảm ơn tất cả các thầy cô trong trường Đại học Công nghệ - Đại học Quốc gia Hà Nội đã tạo điều kiện thuận lợi cho tôi trong suốt thời gian học tập và phát triển tại trường.

Lời cam đoan

Tôi là Công Nghĩa Hiếu, sinh viên lớp QH-2021-I/CQ-I-IT2 khóa K66 theo học ngành Công nghệ thông tin tại trường Đại học Công Nghệ - Đại học Quốc gia Hà Nội. Tôi xin cam đoan khoá luận Xây dựng công cụ phân tích mã nguồn cho ngôn ngữ Rust là công trình nghiên cứu do bản thân tôi thực hiện. Các nội dung nghiên cứu, kết quả trong khoá luận là xác thực.

Các thông tin sử dụng trong khoá luận là có cơ sở và không có nội dung nào sao chép từ các tài liệu mà không ghi rõ trích dẫn tham khảo. Tôi xin chịu trách nhiệm về lời cam đoan này.

Hà Nội, ngày 16 tháng 12 năm 2024

Sinh viên

Công Nghĩa Hiếu

Tóm tắt

Với sự phát triển mạnh mẽ của API, kiểm thử API là một chủ đề được quan tâm đặc biệt trong lĩnh vực nghiên cứu hiện nay, bởi vì API đóng vai trò quan trọng trong việc kết nối các hệ thống và ứng dụng. Đi cùng sự phát triển nhanh chóng của công nghệ và nhu cầu tích hợp liên tục, việc đảm bảo chất lượng và hiệu suất của các API trở nên cấp thiết.

Khóa luận này đã trình bày công cụ kiểm thử tự động API ứng dụng học tăng cường với một phương pháp đánh giá điểm thưởng tập trung vào việc tối ưu hóa mức độ phủ trong kiểm thử API, đảm bảo tính hiệu quả và toàn diện trong quá trình đánh giá. Qua quá trình thực nghiệm trên một tập các dịch vụ kiểm thử uy tín, kết quả cho thấy công cụ được phát triển có thể xác định được 126,9 lỗi riêng biệt với mã trả về dạng `500`, vượt trội hơn 14 lỗi so với ARAT-RL, công cụ kiểm thử tự động API mới nhất hiện nay [11]. Về tốc độ, tại đa số thời điểm, công cụ được trình bày đều cho thấy hiệu suất tốt hơn so ARAT-RL. Đặc biệt, tại một số thời điểm, trên cùng một mốc yêu cầu, công cụ này vượt trội hơn với 15 lỗi tìm được nhiều hơn. Những kết quả này chứng minh rằng công cụ không chỉ cải thiện về mặt số lượng lỗi phát hiện được mà còn tăng tốc độ phát hiện lỗi, giúp tiết kiệm tài nguyên và nâng cao chất lượng phần mềm.

Từ khóa: Phân tích mã nguồn, phân tích tĩnh, ngôn ngữ lập trình Rust

Abstract

With the rapid development of APIs, API testing has become a topic of particular interest in current research because APIs play a crucial role in connecting systems and applications. As technology evolves and the need for continuous integration grows, ensuring the quality and performance of APIs becomes essential.

This thesis presents an automated API testing tool that employs reinforcement learning with a reward evaluation method focused on optimizing test coverage, ensuring efficiency and comprehensiveness in the evaluation process. Through experiments on reputable testing services, the results demonstrate that the developed tool can identify 126.9 distinct errors with a 500 response code, surpassing ARAT-RL, the latest automated API testing tool, by 14 errors. In terms of speed, the presented tool generally shows better performance than ARAT-RL. Notably, at certain points, on the same request milestone, this tool outperforms with 15 more errors detected. These results prove that the tool not only improves the number of detected errors but also increases the speed of error detection, saving resources and enhancing software quality.

Keywords: Source code analysis, static analysis, Rust programming language

Mục lục

Danh sách hình vẽ

Danh sách bảng

Danh mục các từ viết tắt

STT	Từ viết tắt	Cụm từ đầy đủ	Cụm từ tiếng Việt
1	JSON	JavaScript Object Notation	Ký hiệu đối tượng JavaScript
2	OAS	OpenAPI Specification	Đặc tả OpenAPI
3	URL	Uniform Resource Locator	Định vị tài nguyên thống nhất
4	SUT	System Under Test	Hệ thống được kiểm thử
5	HTTP	Hypertext Transfer Protocol	Giao thức truyền tải siêu văn bản
6	YAML	Yet Another Markup Language	Ngôn ngữ đánh dấu YAML
7	REST	Representational State Transfer	Kiến trúc thiết kế REST
8	API	Application Programming Interface	Giao diện lập trình ứng dụng
9	CLI	Command Line Interface	Giao diện dòng lệnh
10	JVM	Java Virtual Machine	Máy ảo Java

Chương 1

Đặt vấn đề

Trong thế giới kỹ thuật hiện đại, việc trao đổi dữ liệu giữa các ứng dụng và hệ thống là một phần không thể thiếu của cuộc sống hàng ngày. Từ việc gửi tin nhắn qua các ứng dụng trò chuyện cho đến việc truy vấn thông tin từ cơ sở dữ liệu trực tuyến, mọi thứ đều dựa vào giao tiếp hiệu quả giữa các phần mềm. Trong đó, Application Programming Interface, viết tắt là API, tạm dịch là "Giao diện lập trình ứng dụng", là một kĩ thuật cho phép liên lạc và trao đổi dữ liệu giữa hai hệ thống phần mềm riêng biệt. API đóng vai trò như một cầu nối, cho phép các ứng dụng và dịch vụ khác nhau giao tiếp và làm việc cùng nhau một cách hiệu quả. Nó là tập hợp các quy tắc, cách thức cho phép hai phần mềm có thể trao đổi thông tin. Cụ thể hơn, API định nghĩa cách các yêu cầu được khởi tạo, cách các yêu cầu hoạt động và định dạng dữ liệu đi kèm để tạo lập kết nối giữa các thành phần trong hệ thống.

Trong số các kiến trúc thiết kế API, REST (Representational State Transfer) là một kiến trúc phổ biến được sử dụng, đã được giới thiệu lần đầu vào năm 2000 [7]. REST được tạo ra với mục đích cung cấp khả năng tương tác với các nguồn tài nguyên mạng một cách đơn giản và hiệu quả. Kiến trúc này dựa trên các giao thức chuẩn như HTTP, URL, và JSON. REST đã trở thành một tiêu chuẩn quan trọng trong phát triển các ứng dụng web hiện đại, giúp tạo ra các API linh hoạt, dễ bảo trì và dễ mở rộng.

Với ngữ cảnh phát triển ứng dụng web, RESTful API được định nghĩa là một API được xây dựng dựa trên kiến trúc REST. Theo Alberto Martin-Lopez và các cộng sự, RESTful API đang phát triển nhanh chóng như một chìa khóa cho việc tái sử dụng mã nguồn, tích hợp và phát triển phần mềm [16]. Các công ty như Facebook, Twitter, Google, eBay hoặc Netflix nhận được hàng tỷ lời gọi API mỗi ngày từ hàng nghìn ứng dụng và thiết bị của bên thứ ba khác nhau, chiếm hơn một nửa tổng lưu lượng của họ [10]. Nói chung, với xu hướng các hệ thống được xây dựng trên kiến trúc microservices, việc sử dụng RESTful API ngày càng trở nên phổ biến [18].

Đi kèm với sự phát triển của RESTful API, việc đảm bảo chất lượng cho RESTful API cũng là một lĩnh vực nhận được sự quan tâm lớn đến từ các doanh nghiệp [5]. Đây là quy trình để chắc chắn rằng RESTful API sẽ thỏa mãn các tiêu chí về chất lượng và chức năng. Điều này bao gồm các khía cạnh như: độ chính xác, hiệu suất, bảo mật và khả năng sử dụng. Thông qua những ca kiểm thử, nhà phát triển phần mềm có thể xác định và ngăn chặn các lỗi trước khi phát hành sản phẩm. Một trong những phương pháp phổ biến hiện nay đó là kết hợp giữa kiểm thử thủ công và kiểm thử tự động. Kiểm thử thủ công là loại kiểm thử phần mềm trong đó các ca kiểm thử được thực hiện thủ công bởi con người, trong khi kiểm thử tự động là quá trình sử dụng các công cụ để tự động hóa quá trình kiểm thử [9].

Việc nghiên cứu về các kĩ thuật trong kiểm thử tự động RESTful API luôn là chủ đề được sự quan tâm đến từ các nhà nghiên cứu. Bằng chứng là sự ra đời của nhiều kỹ thuật tự động mới trong những năm gần đây [5]. Xu hướng nghiên cứu gần đây tập trung vào việc phát triển các phương pháp kiểm thử tự động mới, nhằm nâng cao hiệu quả và tính toàn diện của quá trình kiểm thử phần mềm. Các phương pháp này được kỳ vọng sẽ giúp phát hiện những lỗi tiềm ẩn và khó tìm thấy mà con người có thể bỏ sót.

EvoMaster là công cụ mã nguồn mở tiên phong ứng dụng trí tuệ nhân tạo để tự động tạo ca kiểm thứ cho ứng dụng web [2]. Nó cung cấp hai chế độ kiểm thử: hộp trắng và hộp đen, trong đó chế độ hộp trắng nổi bật hơn nhờ khả năng ứng dụng trí tuệ nhân tạo nhằm tối ưu hóa độ phủ mã nguồn. RESTler là công cụ kiểm thử mờ RESTful API theo hướng kiểm thử hộp đen có trạng thái đầu tiên được phát triển [3]. Trong ngữ cảnh này, thuật ngữ "trạng thái" đề cập đến khả năng khám phá các trạng thái của dịch vụ thông qua việc sinh chuỗi các lời gọi API. Tương tự với RESTler, Morest là một công cụ kiểm thử API hộp đen có trạng thái, hướng tới việc sinh ra các chuỗi yêu cầu [15]. Mới nhất hiện nay, ARAT-RL là một công cụ kiểm thử tự động API được phát triển mới đây, tận dụng học tăng cường để tạo ra các ca kiểm thử một cách thích nghi [11]. Khác với các phương pháp khác, ARAT-RL sáng tạo ở chỗ, đã xác định trọng số của các tham số và thao tác (operation), đồng thời điều chỉnh các trọng số này dựa trên thông tin nhận được từ phản hồi xuyên suốt quá trình kiểm thử. Dựa trên những trọng số đó, công cụ sẽ lựa chọn thứ tự kiểm thử các thao tác và tham số, nhằm nâng cao hiệu suất kiểm thử.

Tuy nhiên, các công cụ hiện có thường tập trung vào việc tạo ra các yêu cầu kiểm thử mà chưa chú trọng việc phân tích và đánh giá phản hồi trả về. Lấy ví dụ, ARAT-RL chỉ sử dụng thông tin mã trạng thái phản hồi, bỏ qua các dữ liệu hữu ích khác. Tương tự, Morest và RESTler chỉ tập trung vào việc tạo chuỗi yêu cầu mà chưa quan tâm đến việc nâng cao hiệu quả phủ đầu ra.

Để giải quyết các điểm yếu của các giải pháp trước đó, khóa luận này trình bày công cụ kiểm thử tự động API ứng dụng học tăng cường dựa trên ARAT-RL. Công cụ trích xuất và xử lý nhiều thông tin và xử lý thông tin hiệu quả hơn, nhằm tối đa hóa sự đa dạng đầu ra, định hướng việc sinh dữ liệu đầu vào trong quá trình đánh giá và sinh các ca kiểm thử. Qua quá trình thực nghiệm, kết quả thử nghiệm cho thấy công cụ vượt qua giải pháp số một hiện nay (ARAT-RL) về số lượng lỗi và tốc độ tìm lỗi. Về mặt số lỗi tìm được, công cụ được của có thể xác định được trung bình 126,9 lỗi riêng biệt với mã trả về có dạng là `500`, vượt trội hơn 14 lỗi so với ARAT-RL. Về tốc độ, tại đa số thời điểm, công cụ được trình bày đều cho thấy hiệu suất tốt hơn so ARAT-RL. Đặc biệt, tại một số thời điểm, trên cùng một mốc yêu cầu, công cụ này vượt trội hơn với 15 lỗi tìm được nhiều hơn.

Khóa luận sẽ được trình bày theo cấu trúc như sau. **Chương 2** trình bày sơ lược một số nền tảng lý thuyết sẽ được sử dụng trong các phần sau của khóa luận. **Chương 3** đề cập chi tiết từng bước của phương pháp được đề xuất trong khóa luận. Phần báo cáo và đánh giá kết quả thực nghiệm được trình bày ở **Chương 4**. Cuối cùng, **Chương 5** trình bày kết luận và đưa ra một số định hướng cải tiến cho giải pháp trong tương lai.

Chương 2

Kiến thức cơ sở

Trong chương này, một số kiến thức chung về Restful API sẽ được giới thiệu. Bên cạnh đó, những kiến thức về kiểm thử API và kiểm thử mờ được đề cập để làm nền tảng cho giải pháp sẽ được đưa ra trong khóa luận này. Phần tiếp theo sẽ tập trung trình bày về học tăng cường và thuật toán Q-learning, một thuật toán được sử dụng rộng rãi trong lĩnh vực này. Phần cuối cùng sẽ trình bày về ARAT-RL, công cụ kiểm thử API mới nhất hiện nay ứng dụng học tăng cường.

2.1 Restful API

Restful API hay còn được gọi là REST API, là các API trên nền trình duyệt web tuân theo phong cách kiến trúc REpresentational State Transfer (REST) [7]. Chúng cho phép người dùng tương tác với các dịch vụ web bằng cách gửi các yêu cầu HTTP và nhận phản hồi. Người dùng sử dụng các yêu cầu để truy cập và thao tác với các tài nguyên được dịch vụ quản lý. Tài nguyên là một phần dữ liệu mà người dùng có thể thực hiện các hành động: tạo, đọc, cập nhật hoặc xóa. Yêu cầu từ người dùng sẽ được gửi đến một điểm truy cập API, đây là đường dẫn tài nguyên xác định tài nguyên, cùng với phương thức HTTP chỉ định hành động được thực hiện trên tài nguyên. Các phương thức phổ biến nhất là *POST, GET, PUT và DELETE*, ứng với lần lượt các hành động người dùng.

Sau khi máy chủ tiếp nhận và xử lý yêu cầu từ người dùng, phản hồi trả về có định dạng gồm 3 phần: tiêu đề (header), nội dung yêu cầu (request body) và mã trạng thái HTTP (HTTP status code). Tiêu đề cung cấp thông tin bổ sung về ngữ cảnh của yêu cầu vừa được thực hiện, như định dạng dữ liệu, thời gian, kiểu mã hóa, v.v. Nội dung yêu cầu chứa dữ liệu chính mà người dùng quan tâm. Mã trạng thái HTTP thể hiện tình trạng hoàn thành của yêu cầu. Mã trạng thái này được chia thành 5 loại:

- Mã `1xx`: Chỉ ra yêu cầu đã được nhận và đang được xử lý.

- Mã `2xx`: Cho biết yêu cầu đã được xử lý thành công. Trong nhóm này, mã `200` là mã thành công phổ biến nhất.
- Mã `3xx`: Yêu cầu chuyển hướng, ví dụ như tài nguyên mục tiêu đã được di chuyển đến một URL mới (mã `301`).
- Mã `4xx`: Chỉ ra lỗi của khách hàng. Ví dụ, mã `400` biểu thị yêu cầu không hợp lệ, thường do giá trị đầu vào không chính xác, trong khi mã `404` cho biết khách hàng đã yêu cầu một tài nguyên không tồn tại.
- Mã `5xx`: Chỉ ra lỗi của máy chủ trong quá trình xử lý yêu cầu. Đặc biệt, mã `500` cho biết lỗi máy chủ nội bộ và thường xảy ra trong các trường hợp dịch vụ gặp lỗi và không thể xử lý yêu cầu một cách chính xác. Do đó, nhiều nghiên cứu thực nghiệm về các công cụ kiểm thử REST API, bao gồm cả báo cáo này, xem xét số lượng mã trạng thái `500` độc nhất được tìm thấy như là các lỗi của dịch vụ.

RESTful API được xây dựng dựa trên giao thức HTTP, nên mang theo đặc tính quan trọng là phi trạng thái. Điều này có nghĩa là mỗi yêu cầu từ người dùng chỉ cần cung cấp đầy đủ thông tin để máy chủ hiểu và xử lý, mà không phụ thuộc vào bất kỳ trạng thái nội bộ nào trên máy chủ. Đặc tính phi trạng thái này giúp đơn giản hóa và nâng cao hiệu quả cho giao thức RESTful API, do máy chủ không cần quản lý trạng thái, từ đó giảm thiểu độ phức tạp của hệ thống. Bên cạnh đó, việc tận dụng tính chất phi trạng thái cũng góp phần tăng khả năng mở rộng và tương thích cho các hệ thống sử dụng RESTful API. Nhờ những ưu điểm này, RESTful API ngày càng được ưa chuộng và thường được triển khai trong các ứng dụng web, di động, hệ thống phân tán và Internet of Things (IoT) [4, 17].

```
import numpy as np

def incmatrix(genl1, genl2):
    m = len(genl1)
    n = len(genl2)
    M = None  # to become the incidence matrix
    VT = np.zeros((n*m, 1), int)  # dummy variable

    # compute the bitwise xor matrix
    M1 = bitxormatrix(genl1)
```

```
    M2 = np.triu(bitxormatrix(genl2), 1)

    for i in range(m-1):
        for j in range(i+1, m):
            [r, c] = np.where(M2 == M1[i, j])
            for k in range(len(r)):
                VT[(i)*n + r[k]] = 1
                VT[(i)*n + c[k]] = 1
                VT[(j)*n + r[k]] = 1
                VT[(j)*n + c[k]] = 1

            if M is None:
                M = np.copy(VT)
            else:
                M = np.concatenate((M, VT), 1)

            VT = np.zeros((n*m, 1), int)

    return M
```

```
import numpy as np

def incmatrix(genl1, genl2):
    m = len(genl1)
    n = len(genl2)
    M = None # to become the incidence matrix
    VT = np.zeros((n*m, 1), int) # dummy variable

    # compute the bitwise xor matrix
    M1 = bitxormatrix(genl1)
    M2 = np.triu(bitxormatrix(genl2), 1)

    for i in range(m-1):
        for j in range(i+1, m):
            [r, c] = np.where(M2 == M1[i, j])
            for k in range(len(r)):
                VT[(i)*n + r[k]] = 1
                VT[(i)*n + c[k]] = 1
                VT[(j)*n + r[k]] = 1
                VT[(j)*n + c[k]] = 1

            if M is None:
```

```
            M = np.copy(VT)
        else:
            M = np.concatenate((M, VT), 1)

        VT = np.zeros((n*m, 1), int)

  return M
```

2.2 Đặc tả OpenAPI

```
openapi: 3.0.0
info:
  title: Simple Pet API
  version: 1.0.0
  description: A simple API for managing pets
servers:
  - url: https://petstore.swagger.io/v2
paths:
  /pets/{id}:
    get:
      summary: Get a pet by id
      parameters:
        - name: id
          in: path
          required: true
          schema:
            type: integer
            format: int64
      responses:
        '200':
          description: The pet object
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Pet'
        '404':
          description: Pet not found
components:
```

```
schemas:
  Pet:
    type: object
    properties:
      id:
        type: integer
        format: int64
      name:
        type: string
      tag:
        type: string
```

Hình 2.1: Đặc tả OpenAPI

- **Điểm truy cập:** Địa chỉ truy cập của API.
- **Thao tác:** Các hành động có thể thực hiện trên API (ví dụ: GET, POST, PUT, DELETE).
- **Định dạng yêu cầu và phản hồi:** Cấu trúc dữ liệu của yêu cầu gửi lên API và dữ liệu trả về từ API.
- **Tham số:** Các biến số được sử dụng trong yêu cầu gửi lên API.
- **Phương thức xác thực:** Cách thức xác minh người dùng khi truy cập API.

Mục tiêu chính của OAS là xác định một định dạng chuẩn, không phụ thuộc ngôn ngữ để mô tả RESTful API, cho phép cả con người và máy tính khám phá và hiểu các khả năng của dịch vụ mà không cần truy cập vào mã nguồn. Đặc tả OAS sau đó có thể được sử dụng bởi các công cụ tạo tài liệu để hiển thị API, các công cụ kiểm thử và nhiều trường hợp sử dụng khác.

2.3 Kiểm thử API

Để đảm bảo độ tin cậy của API, kiểm thử là một phương pháp thông thường và quan trọng trong quy trình phát triển API [9]. Đây là một loại kĩ thuật kiểm tra tính hợp lệ của API. Mục đích của kĩ thuật này là kiểm tra chức năng, độ tin cậy, hiệu suất và bảo mật của các API.

Với sự phát triển của lĩnh vực này, việc tự động hóa cho kiểm thử API đang trở thành một chủ đề nghiên cứu ngày càng được quan tâm. Kĩ thuật này thay vì sử dụng đầu vào và đầu ra tiêu chuẩn như bàn phím, người dùng sẽ sử dụng công cụ để tự động hóa quá trình gửi các lời gọi API, xử lý đầu ra và phản hồi của hệ thống. Kĩ thuật chủ yếu tập trung vào lớp logic nghiệp vụ của kiến trúc phần mềm.

Nghiên cứu về kiểm thử tự động API có thể được chia thành hai phương pháp: kiểm thử hộp đen và kiểm thử hộp trắng. Phương pháp kiểm thử hộp đen sinh dữ liệu kiểm thử dựa trên thông tin từ đặc tả API, thường ở dạng OpenAPI [**?**]. Phương pháp kiểm hộp trắng sử dụng cả đặc tả API và mã nguồn chương trình để tạo các ca kiểm thử bao phủ tốt nhất có thể, cố gắng giảm thiểu khối lượng công việc thủ công từ người dùng. Cả hai phương pháp đều cố gắng tự động xác định lỗi máy chủ (ví dụ: mã trạng thái HTTP `5xx`) và các sự mâu thuẫn của phản hồi so với đặc tả API. Với mục tiêu này, quá trình kiểm thử tự động không chỉ giúp đảm bảo tính đúng đắn và ổn định của API mà còn giúp tăng hiệu suất và giảm thiểu chi phí con người trong quá trình phát triển phần mềm.

2.3.1 Mục tiêu độ phủ

Trong kiểm thử phần mềm nói chung và cụ thể kiểm thử API, để đánh giá chất lượng của các ca kiểm thử được tạo, ta có 3 thang đo phổ biến sau: độ phủ đầu vào, độ phủ đầu ra và độ phủ mã nguồn. Mỗi thang đo sẽ đánh giá một khía cạnh khác nhau của bộ kiểm thử.

Độ phủ đầu vào thể hiện cho mức độ đa dạng mà tập các ca kiểm thử có thể kiểm thử trên tập đầu vào của chương trình máy tính. Tiêu chí này đóng vai trò quan trọng trong lĩnh vực kiểm thử, thể hiện trong cả lĩnh vực kiểm thử API. Các API dựa vào các đầu vào để có thể hoạt động chính xác. Việc không kiểm thử các đầu vào có thể xảy ra, hợp lệ hoặc bất thường, có thể để lại lỗ hổng bảo mật nghiêm trọng. Độ phủ đầu vào giúp đảm bảo API phản hồi như mong đợi đối với dữ liệu chuẩn và xử lý các trường hợp bất thường một cách chắc chắn.

Độ phủ đầu ra tập trung vào khả năng của bộ kiểm thử có thể phủ tất cả các phản hồi có thể có của API. Nói cách khác, phản hồi càng đa dạng thì độ phủ đầu ra càng cao. Trong kiểm thử API, độ phủ đầu ra liên quan đến độ phủ mã trạng thái.

Ngoài ra, các phản hồi cũng được phân biệt theo nội dung đi kèm. Độ phủ đầu ra giúp đánh giá tính đầy đủ và tính chính xác của chức năng của hệ thống bằng cách đánh giá các dạng đầu ra khác nhau mà hệ thống trả về.

Độ phủ mã nguồn đo lường mức độ mã nguồn của API được thực thi trong quá trình kiểm thử. Nó chỉ ra tỷ lệ các dòng mã, nhánh, câu lệnh hoặc điều kiện đã được bộ kiểm thử thực thi. Theo đó, có một số loại độ phủ mã nguồn. Độ phủ dòng có nhiệm vụ đo lường tỷ lệ các dòng mã đã được thực thi so với tổng số dòng mã trong mã nguồn. Độ phủ câu lệnh đo lường tỷ lệ các dòng mã đã được thực thi so với tổng số dòng mã trong mã nguồn. Độ phủ câu lệnh đo lường tỷ lệ các dòng mã đã được thực thi so với tổng số dòng mã trong mã nguồn. Độ phủ điều kiện đo lường tỷ lệ các điều kiện logic trong câu lệnh đã được thực thi.

Trong quá trình kiểm thử, việc đạt độ phủ cao là một trong những mục tiêu của các bộ kiểm thử. Điều này thể hiện rằng, nó đã bao quát được nhiều phần của hệ thống được kiểm thử, có thể giúp tìm ra được nhiều lỗi tiềm ẩn của hệ thống. Tuy nhiên, việc tập trung quá mức vào việc tối ưu hóa độ phủ có thể dẫn đến việc tạo ra các bài kiểm thử không hiệu quả, bỏ sót các lỗi tiềm ẩn nằm trong các phần đã được phủ. Do đó, mục tiêu độ phủ chỉ nên được xem như một khía cạnh trong quá trình kiểm thử, cần kết hợp với các phương pháp kiểm thử khác để đảm bảo hiệu quả và tính toàn diện của quá trình kiểm thử.

2.3.2 Thách thức trong kiểm thử API

Kiểm thử API đóng vai trò quan trọng trong việc đảm bảo tính nhất quán và chức năng của các dịch vụ web. Tuy nhiên, quá trình này gặp phải nhiều thách thức lớn. Đầu tiên, phạm vi kiểm thử rất rộng và phức tạp do API cung cấp nhiều điểm truy cập với các chức năng khác nhau, đòi hỏi phải kiểm tra kỹ lưỡng từng điểm để đảm bảo API hoạt động đúng yêu cầu. Hơn nữa, việc xử lý dữ liệu đầu vào và đầu ra là một thách thức khác, bởi API thường phải xử lý nhiều loại dữ liệu khác nhau như JSON, XML... Điều này yêu cầu sự hiểu biết về cấu trúc dữ liệu và các tình huống sử dụng. Tính ổn định của môi trường kiểm thử cũng là một vấn đề, bởi các yếu tố như sập máy chủ, cơ sở dữ liệu thay đổi, hoặc dịch vụ phụ thuộc không hoạt động có thể gây khó khăn cho việc kiểm thử chính xác. Mặt khác, vấn đề bảo mật cần được chú trọng đặc biệt để đảm bảo API không bị tấn công bởi các phương pháp như SQL Injection, XSS, và DDoS. Kiểm thử hiệu suất và chịu

tải cũng rất quan trọng để đảm bảo API có thể xử lý lượng lớn yêu cầu trong thời gian ngắn. Cuối cùng, các API thường phụ thuộc vào dịch vụ bên thứ ba hoặc các API khác, do đó việc kiểm thử phải đảm bảo tất cả các tích hợp hoạt động mượt mà. Những thách thức này đòi hỏi các kỹ sư kiểm thử phải có kỹ năng kỹ thuật cao, hiểu biết sâu rộng về hệ thống và khả năng sử dụng các công cụ kiểm thử tự động một cách hiệu quả.

2.4 Kiểm thử mờ

Kiểm thử mờ là một kỹ thuật kiểm thử tự động nhằm bao phủ nhiều trường hợp biên bằng cách sử dụng dữ liệu không hợp lệ (từ tệp, giao thức mạng, các API) làm đầu vào của ứng dụng để đảm bảo phủ tốt nhất các lỗi có thể [14]. Ý tưởng chính đằng sau kĩ thuật là tạo ra và kiểm thử nhiều ca kiểm thử nhằm tìm ra các lỗi phần mềm khó gặp. Các phương pháp tạo dữ liệu đầu vào có thể được chia thành hai loại chính: kiểm thử mờ hộp đen và kiểm thử mờ hộp trắng.

Kiểm thử mờ hộp đen sở hữu những đặc tính quan trọng của kiểm thử hộp đen truyền thống. Các dữ liệu đầu vào được sinh ra mà không cần thông tin về cấu trúc bên trong của ứng dụng. Phương pháp này đơn giản và dễ triển khai, nhưng có thể bỏ sót các lỗi phức tạp do không khai thác được kiến trúc nội bộ của ứng dụng. Để hạn chế những điểm yếu này, kỹ thuật kiểm thử mờ sử dụng những kỹ thuật kiểm thử hộp đen tiên tiến hơn như nhóm các thuật toán di truyền (genetic algorithms). Các thuật toán này mô phỏng quá trình chọn lọc tự nhiên để phát triển dữ liệu đầu vào tốt hơn theo thời gian. Bằng cách lai ghép và đột biến dữ liệu đầu vào hiện có, chúng giúp tạo ra các đầu vào mới có khả năng cao gây ra lỗi.

Kiểm thử mờ hộp trắng tận dụng thông tin về mã nguồn để tạo ra các ca kiểm thử. Kỹ thuật này thường kết hợp với các phương pháp phân tích mã nguồn để tối ưu hóa dữ liệu đầu vào và nâng cao độ phủ kiểm thử. Nhờ vậy, kiểm thử mờ hộp trắng có thể tìm ra nhiều lỗi hơn và chính xác hơn so với kiểm thử mờ hộp đen. Một trong những chiến lược chính của kỹ thuật này là chiến lược sinh dữ liệu được định hướng bởi mục tiêu độ phủ mã nguồn. Chiến lược này sử dụng thông tin về độ phủ mã nguồn để định hướng việc tạo ra dữ liệu đầu vào mới với mục tiêu là tăng độ phủ của kiểm thử. Bằng cách này, kỹ thuật kiểm thử mờ hộp trắng giúp phát hiện ra nhiều lỗi hơn, cải thiện chất lượng và độ tin cậy của phần mềm.

Mục tiêu quan trọng nhất của kiểm thử mờ là cách tạo ra các ca kiểm thử phù hợp, khai thác các phương pháp được trình bày ở trên để giúp tìm ra các lỗi ẩn trong hệ thống. Đây là một phương pháp kiểm thử hiện đại, đẩy mạnh khả năng tự động hóa, giúp giảm thiểu sự can thiệp của con người và tăng hiệu quả kiểm thử.

2.5 Học tăng cường

Học tăng cường là một nhánh của học máy tập trung vào việc huấn luyện một tác nhân để đưa ra các quyết định tối ưu bằng cách tương tác với môi trường [19]. Hình **??** cho ta thấy luồng hoạt động tổng quát của học tăng cường [**?**]. Tác nhân lựa chọn hành động trong các tình huống khác nhau (trạng thái), quan sát kết quả và học cách chọn hành động tốt nhất để tối đa hóa tổng điểm thưởng theo thời gian. Quá trình học trong học tăng cường dựa trên phương pháp thử nghiệm và sai sót (trial and error), nghĩa là tác nhân học được hành động tốt nhất bằng cách thử nghiệm với các lựa chọn khác nhau và cập nhật chiến lược của mình dựa trên phản hồi. Tác nhân cũng phải cân bằng giữa việc khám phá các hành động mới để có thêm thông tin hoặc khai thác các hành động đã biết mang lại điểm thưởng tốt nhất dựa trên kiến thức hiện tại của nó. Sự thay đổi giữa khám phá thông tin mới và khai thác những thông tin đã có thường được kiểm soát bởi các tham số, chẳng hạn như ϵ trong chiến lược ϵ-greedy [19].

Q-learning là một thuật toán phổ biến trong học tăng cường để tác nhân học cách đưa ra hành động tối ưu trong một môi trường. Nó tính xấp xỉ hàm Q-function (Q (s, a)), tính toán điểm thưởng dài hạn dự kiến khi thực hiện một hành động trong một trạng thái nhất định và sau đó lựa chọn chính sách (policy) tốt nhất. Q-learning sử dụng một bảng 2 chiều Q-table để lưu trữ và cập nhật các giá trị Q-value dựa trên sự tương tác của tác nhân với môi trường. Tác nhân khám phá môi trường, nhận phản hồi và cập nhật các giá trị Q-value theo quy tắc cập nhật Q-learning, được bắt nguồn từ phương trình Bellman [19]:

trong đó α là tốc độ học, γ là hệ số chiết khấu (discount factor), $s^{'}$ là trạng thái mới sau khi thực hiện hành động α và r là điểm thưởng tức thời nhận được. Tác nhân cập nhật các giá trị Q-value để hội tụ về các giá trị tối ưu của chúng, đại diện cho điểm thưởng dài hạn dự kiến của việc thực hiện từng hành động trong

từng trạng thái.

2.6 Các công cụ kiểm thử hiện thời

2.6.1 Evomaster

EvoMaster là một công cụ mã nguồn mở được biết đến là công cụ kiểm thử đầu tiên sử dụng trí tuệ nhân tạo để tự động sinh các ca kiểm thử cho ứng dụng mạng. Công cụ có thể sinh các ca kiểm thử API cho REST, GraphQL và RPC [8, 13]. Hơn nữa, EvoMaster cung cấp hai chế độ kiểm thử: kiểm thử hộp trắng hoặc kiểm thử hộp đen. Cả hai chế độ đều bắt đầu bằng việc xử lý đặc tả OpenAPI đầu vào để lấy thông tin cho quá trình kiểm thử.

Đối với kiểm thử hộp trắng, chế độ này yêu cầu quyền truy cập vào mã nguồn chương trình và cũng là chế độ mang lại hiệu quả vượt trội của Evomaster. Nó sử dụng giải thuật tiến hóa (theo mặc định là thuật toán MIO) để tạo ra các ca kiểm thử với mục tiêu tối đa hóa độ phủ mã nguồn [1]. Cụ thể, đối với mỗi nhánh chưa được phủ, công cụ phát triển các ca kiểm thử bằng cách liên tục tạo ra ca kiểm thử mới đồng thời loại bỏ những ca kiểm thử ít khả quan nhất để thực hiện kiểm thử trên nhánh đó lặp lại cho đến khi đạt đến một mốc giới hạn thời gian. Kĩ thuật trong chế độ này tạo các kiểm thử mới thông qua việc lấy mẫu (sampling) hoặc biến đổi từ các ca kiểm thử trước đó (mutation). Ngược lại, trong chế độ kiểm thử hộp đen, Evomaster tập trung vào kĩ thuật kiểm thử mờ, cung cấp đầu vào cho việc kiểm thử bằng cách sinh ngẫu nhiên các trường hợp không hợp lệ hoặc hiếm gặp. Tuy nhiên, kết quả kiểm thử hộp đen có thể không hiệu quả bằng kiểm thử hộp trắng do thiếu phân tích mã nguồn. Điểm chung của hai phương pháp này là đều xem xét việc tìm kiếm các lỗi `5xx` bên phía máy chủ.

2.6.2 RESTler

RESTler là công cụ kiểm thử mờ REST API có trạng thái đầu tiên được phát triển bởi Microsoft [3]. Trong ngữ cảnh này, thuật ngữ "trạng thái" đề cập đến khả năng khám phá các trạng thái của dịch vụ mà chỉ có thể truy cập được thông qua chuỗi các lời gọi API được gửi đến hệ thống. Hình **??** mô tả luồng hoạt động cơ bản của RESTler. Công cụ nhận đầu vào là một đặc tả OpenAPI (với tên gọi cũ là đặc tả

Swagger), tiến hành xử lý đặc tả và dịch sang "RESTler Grammar", tạm dịch là ngữ pháp RESTler. Hình **??** cho ta một ví dụ, với bên trái là đặc tả OpenAPI mô tả về một thao tác và bên phải là ngữ pháp RESTler tương ứng được tự động sinh bởi công cụ. Dựa trên ngữ pháp này, RESTler tạo ra các chuỗi yêu cầu bằng cách suy luận các phụ thuộc giữa các thao tác trong đặc tả API và phân tích các phản hồi được ghi lại trong các lần thực hiện các chuỗi trước đó để tạo các chuỗi mới. Công cụ này tính toán tất cả các chuỗi yêu cầu hợp lệ với độ dài tăng dần, cho đến khi các chuỗi đạt đến độ dài được xác định trước. Một chuỗi được coi là hợp lệ khi tất cả các yêu cầu trong chuỗi đó thành công và nhận mã trả về dạng `2xx` (thành công).

Tuy nhiên, vì RESTler chỉ tập trung vào việc sinh các chuỗi, các giá trị đầu vào của công cụ có thể không bao quát được nhiều các tình huống kiểm thử khả thi hoặc các trường hợp ngoại lệ, dẫn đến một số khía cạnh của API không được kiểm thử. Ngoài ra, mặc dù đã phân tích các phản hồi, nhưng nó không sử dụng thông tin này để tạo các ca kiểm thử nhằm ưu tiên tìm ra lỗi. Những hạn chế này khiến bộ kiểm thử của nó không đạt được độ phủ đầu ra và độ phủ mã nguồn mong muốn.

2.6.3 Morest

Morest là một công cụ kiểm thử API có trạng thái, dựa trên việc tạo ra đồ thị Thuộc tính Dịch vụ RESTful (RPG) với các phụ thuộc được trích xuất giữa các API [15]. Đồ thị RPG mô tả chi tiết các phụ thuộc của API và cho phép tinh chỉnh các phụ thuộc đã được ghi lại được một cách linh hoạt trong quá trình kiểm thử. Trong RPG, mỗi nút biểu thị một thao tác hoặc một lược đồ, và cạnh biểu thị các phụ thuộc dữ liệu giữa chúng.

Hình **??** cho thấy luồng hoạt động chi tiết của Morest. Để kiểm thử một dịch vụ RESTful, đầu tiền, công cụ nhận đầu vào là đặc tả OpenAPI của dịch vụ đó làm đầu vào để xây dựng RPG. Sau khi khởi tạo RPG, Morest sử dụng nó để tạo ra các chuỗi lời gọi API được gửi đến dịch vụ RESTful mục tiêu và thu thập các phản hồi. Bằng cách phân tích các phản hồi đã thu thập, Morest lưu lại các yêu cầu thất bại nhằm mục đích phân loại lỗi. Hơn nữa, công cụ cũng sử dụng các phản hồi để tinh chỉnh RPG bằng cách thêm các cạnh còn thiếu và loại bỏ các cạnh không khả thi. RPG đã được tinh chỉnh sau đó tiếp tục được sử dụng để tạo ra nhiều chuỗi

kiểm thử hơn. Đó là điểm đánh dấu kết thúc của một vòng lặp và Morest sẽ tiếp tục kiểm thử và tinh chỉnh RPG cho đến khi đạt đến giới hạn thời gian.

Tuy nhiên, tương tự như RESTler, Morest không chú trọng đến tính đa dạng của đầu vào và đầu ra. Các điều chỉnh động dựa trên RPG của nó chỉ tìm ra các lỗi do sự kết hợp giữa các thao tác, trong khi các tham số cũng có thể gây lỗi theo những cách khác nhau. Morest có thể bỏ sót các tình huống kiểm thử tiềm năng, đặc biệt là các giá trị biên và các trường hợp ngoại lệ. Hơn nữa, việc thiếu tập trung vào đa dạng hóa các giá trị đầu vào làm giảm khả năng phát hiện các lỗi bảo mật hoặc hiệu suất. Những hạn chế này khiến công cụ không tối ưu được hiệu quả kiểm thử.

2.6.4 ARAT-RL

ARAT-RL là một công cụ kiểm thử tự động REST API mới nhất hiện nay, ứng dụng Học tăng cường (Reinforcement Learning). Công cụ này được giới thiệu trong bài báo "Adaptive REST API Testing with Reinforcement Learning" được trình bày tại hội nghị ASE 2023. Trước đó, việc kiểm thử REST API có thể trở nên khó khăn do không gian tìm kiếm rộng lớn cần khám phá. Điều này xuất phát từ nhiều yếu tố như số lượng lớn các thao tác, thứ tự thực thi tiềm năng, sự phụ thuộc giữa các tham số và các ràng buộc giá trị đầu vào của tham số. Các kỹ thuật trước đó thường gặp khó khăn khi khám phá không gian này do thiếu các chiến lược khám phá hiệu quả cho các thao tác và tham số của chúng.

Trên thực tế, trước ARAT-RL, các công cụ kiểm thử hiện có thường đối xử tất cả các thao tác và tham số một cách ngang nhau, bỏ qua việc đánh giá tầm quan trọng hoặc mức độ phức tạp của chúng, dẫn đến các chiến lược kiểm thử không tối ưu và phạm vi phủ sóng không đủ đối với các kết hợp thao tác và tham số quan trọng. Hơn nữa, các công cụ này dựa vào việc khám phá mối quan hệ giữa các lược đồ phản hồi (response schema) và tham số của yêu cầu (resquest parameter). Cách tiếp cận này hoạt động tốt khi các lược đồ tham số và phản hồi được mô tả chi tiết trong lược đồ, nhưng lại không hiệu quả trong trường hợp phổ biến là các đặc tả không đầy đủ hoặc không chính xác.

Để giải quyết vấn về trên, ARAT-RL gán các trọng số cho các thao tác và tham số, ứng dụng học tăng cường để ưu tiên lựa chọn các thao tác và tham số, nhằm tối ưu không gian được kiểm thử. Bên cạnh đó, công cụ cung cấp kĩ thuật phân

tích động để xác định mối quan hệ giữa các tham số và giới thiệu các chiến thuật để xử lý các phản hồi API trước đó để làm dữ liệu cho các ca kiểm thử tiếp theo. Kết quả bài báo cũng cho thấy ARAT-RL đạt kết quả tốt hơn so với các công cụ kiểm thử đối thủ, tạo ra nhiều yêu cầu hợp lệ và gây lỗi hơn, bao phủ nhiều thao tác hơn, đạt được độ phủ mã nguồn cao hơn và tìm được nhiều lỗi dịch vụ hơn.

Luồng hoạt động

Hình **??** cung cấp tổng quan về phương pháp tiếp cận của ARAT-RL, bao gồm hai giai đoạn. Giai đoạn đầu tiên khởi tạo các biến và bảng cần thiết cho quá trình học Q-learning. Giai đoạn tiếp theo liên quan đến việc thực thi thuật toán Q-learning và thu thập thông tin gọi API sau mỗi lần lặp lại từ hệ thống đang thử nghiệm (SUT). Khi hoàn thành, công cụ phân tích bộ sưu tập các cuộc gọi API được tạo ra để xác định các lỗi tiềm ẩn.

Giai đoạn khởi tạo bảng Q-table đóng vai trò quan trọng trong việc thiết lập các cấu trúc thông tin định hướng việc ra quyết định trong suốt quá trình kiểm thử. Cách tiếp cận của ARAT-RL xây dựng trạng thái ban đầu của bảng Q-table liên quan đến việc phân tích tần suất sử dụng các tham số của mỗi thao tác trong đặc tả. Công cụ ưu tiên các tham số xuất hiện thường xuyên vì chúng dường như mang sự quan trọng hơn. Cách tiếp cận này đảm bảo các thao tác cấu thành từ các tham số đó sẽ được ưu tiên kiểm thử đầu tiên. Để tăng sự linh hoạt trong các ca kiểm thử, ARAT-RL cũng đã kết hợp các nguồn đầu vào đa dạng cho các thao tác. Trong giai đoạn khởi tạo, công cụ tạo ra một bảng Q-table để theo dõi điểm hiệu suất của từng nguồn. Điều này giúp ARAT-RL chọn nguồn đầu vào hiệu quả nhất cho các lượt kiểm thử tiếp theo.

Giai đoạn thực thi chịu trách nhiệm triển khai thuật toán Q-Learning. Cách tiếp cận gồm hai chiến lược then chốt: chiến lược ưu tiên dựa trên Q-Learning và chiến lược cập nhật Q-Learning. Mỗi chiến lược sẽ đảm nhiệm từng phần nhỏ trong quá trình này. Chiến lược ưu tiên dựa trên Q-Learning mô tả cách công cụ chọn các thao tác để tiến hành kiểm thử, cùng với các tham số và nguồn đầu vào tương ứng của chúng. Chiến lược cập nhật Q-Learning mô tả cả cơ chế cập nhật bảng Q-table và thúc đẩy quá trình học trong thuật toán Q-Learning.

Trong chiến lược ưu tiên, để tạo đầu vào cho một ca kiểm thử trong kiểm thử

Bảng 2.1: Bảng so sánh 2 công cụ EvoMaster và RESTler

Loại kiểm thử	**EvoMaster**	**RESTler Fuzzer**
Loại kiểm thử	Hộp đen, hộp trắng	Hộp đen
Giao diện	Giao diện dòng lệnh	Giao diện dòng lệnh
Đầu vào	OpenAPI, GraphQL	OpenAPI
Đầu ra	Java Junit 4, 5	File log dạng TXT + JSON
Thời gian chạy	Tùy chỉnh theo giờ (Thời gian chạy thường lâu)	Tùy chỉnh theo giờ (Thời gian chạy thường lâu)
Tái hiện lỗi	Dựa vào file Java Junit để tái hiện thủ công	Đầu ra khó đọc, giảm khả năng tái hiện lỗi
Hạn chế khác	Kết quả không trả về tất cả các ca kiểm thử được thực hiện, mà chỉ có 1 số ca kiểm thử mà công cụ cho là **BUG**	Kết quả khó đọc nên cản trở việc tái hiện lỗi và không hỗ trợ chuyển tiếp yêu cầu

RESTful API, có ba thành phần cơ bản cần được xác định: thao tác, tham số của thao tác và giá trị đầu vào. Những yếu tố này đóng vai trò then chốt trong việc xác định lỗi trong thao tác API đang được kiểm thử. Việc lựa chọn chúng được hướng dẫn bởi tỷ lệ khám phá (ϵ) được duy trì trong suốt giai đoạn học hỏi để đạt được sự cân bằng giữa khai thác và khám phá:

- **Lựa chọn thao tác:** Thao tác API tối ưu được xác định bằng cách lặp qua từng thao tác và tính toán giá trị Q-value trung bình cho các tham số của thao tác đó. Thuật toán chọn thao tác sử dụng một số ngẫu nhiên, được ký hiệu là ϵ, để quyết định liệu thao tác được chọn tiếp theo sẽ nhằm khám phá mới hay khai thác sâu những thông tin đã có.

- **Lựa chọn tham số:** Một thao tác thường bao gồm hai loại tham số: bắt buộc và không bắt buộc (tùy chọn). Chiến lược được xác định để chọn một tập con các tham số tùy chọn cho một thao tác API đã chọn. Tương tự, thuật toán chọn tham số sinh một số ngẫu nhiên để xác định xem những tham số được chọn sẽ nhằm khám phá mới hay khai thác sâu những thông tin đã có. Nếu giá trị được tạo ngẫu nhiên vượt quá ϵ, thuật toán sẽ ưu tiên n các tham số hàng đầu, được xếp hạng theo thứ tự giảm dần dựa trên các giá trị Q-value

tương ứng của chúng, phù hợp với việc ưu tiên đạt được các mục tiêu đã xác định (như đã đề cập trước đó). Ngược lại, một giá trị tùy ý sẽ được sử dụng để chọn ngẫu nhiên n các tham số hàng đầu từ toàn bộ tập tham số của thao tác nếu giá trị đó thấp hơn ϵ. Sự ngẫu nhiên này đảm bảo rằng thuật toán đa dạng hóa các kết hợp tham số, do đó giúp xác định các đầu ra API chưa được kiểm thử trước đó. Cuối cùng, tập tham số được chọn sẽ được trả về.

- **Lựa chọn đầu vào:** Việc lựa chọn các giá trị đầu vào đóng vai trò quan trọng trong việc xác định kết quả của một trường hợp kiểm thử. Giá trị đầu vào của ARAT-RL được chọn dựa trên năm nguồn sau:
  - *Giá trị ví dụ trong tài liệu:* Các giá trị mẫu được cung cấp trong tài liệu hướng dẫn của API.
  - *Giá trị được xây dựng dựa trên phản hồi trước đó:* Sử dụng dữ liệu trả về từ một yêu cầu trước đó để tạo thành giá trị đầu vào cho yêu cầu tiếp theo.
  - *Giá trị được xây dựng dựa trên yêu cầu trước đó:* Sử dụng dữ liệu từ một yêu cầu trước đó để tạo thành giá trị đầu vào cho yêu cầu tiếp theo.
  - *Giá trị mặc định (nếu có):* Sử dụng giá trị mặc định được cung cấp bởi API nếu có.
  - *Giá trị ngẫu nhiên dựa trên các ràng buộc của tài liệu:* Giá trị được tạo ngẫu nhiên nhưng vẫn nằm trong phạm vi cho phép của đặc tả kỹ thuật.

Sau khi hoàn thành việc lựa chọn các thành phần của các ca kiểm thử và thực thi ca kiểm thử đó, ARAT-RL cập nhật bảng Q-table dựa trên phản hồi từ hệ thống, cụ thể là cập nhật các giá trị Q-value dựa trên mã trạng thái của phản hồi. Nếu mã trạng thái trả về thành công (`2xx`), ARAT-RL gán điểm thưởng là -1 và cập nhật các giá trị Q-value theo hướng tiêu cực. Nếu mã trạng thái phản hồi cho thấy yêu cầu không thành công (`4xx` hoặc `500`), thuật toán sẽ gán điểm thưởng là 1 và cập nhật các giá trị Q-value theo hướng tích cực. Các giá trị Q-value được cập nhật cho từng tham số trong các tham số được chọn bằng cách sử dụng quy tắc cập nhật Q-learning, và các giá trị Q-value được cập nhật sẽ được trả về.

Các vấn đề tồn đọng

Mặc dù ARAT-RL là một công cụ kiểm thử RESTful API ứng dụng học tăng cường để mô hình hóa bài toán một cách hiệu quả và thể hiện tiềm năng nhất định, chiến lược cập nhật Q-learning hiện tại (hay còn gọi là chiến lược đánh giá điểm thưởng) vẫn còn bộc lộ nhiều hạn chế trong việc khai thác tối ưu thông tin phản hồi từ hệ thống. Chiến lược này chủ yếu dựa vào thông tin mã trạng thái phản hồi từ API, nhưng lại bỏ qua nhiều dữ liệu hữu ích khác có thể giúp cải thiện quá trình học và tối ưu hóa. Ví dụ, các thông tin chi tiết về cấu trúc phản hồi, tính mới của phản hồi,... đều có thể cung cấp những thông tin quý giá để điều chỉnh và nâng cao hiệu quả của chiến lược học tăng cường. Việc chỉ sử dụng mã trạng thái phản hồi dẫn đến một tầm nhìn hạn hẹp và có thể bỏ qua việc tối ưu hóa độ phủ đầu ra, làm giảm khả năng phát hiện các lỗi tiềm ẩn trong API.

Hơn nữa, hiện tại, ARAT-RL hỗ trợ năm nguồn sinh đầu vào được cài đặt song song. Mặc dù vậy, ARAT-RL có thể được mở rộng và tinh chỉnh các nguồn sinh dữ liệu để gia tăng sự đa dạng cho đầu vào, giúp hệ thống trở nên linh hoạt và hiệu quả hơn trong việc kiểm thử API. Việc mở rộng này có thể bao gồm việc tích hợp thêm các thuật toán sinh ngẫu nhiên tiên tiến, sử dụng dữ liệu từ các nguồn thực tế hơn, hoặc thậm chí là áp dụng các kỹ thuật học máy để tạo ra các mẫu dữ liệu đầu vào phức tạp và có ý nghĩa hơn.

Bằng cách cải tiến chiến lược cập nhật Q-learning và mở rộng các nguồn sinh dữ liệu đầu vào, ARAT-RL không chỉ có thể nâng cao khả năng khai thác thông tin phản hồi mà còn tăng cường khả năng kiểm thử và tối ưu hóa API một cách toàn diện. Điều này sẽ giúp công cụ trở nên mạnh mẽ hơn, đáp ứng được các yêu cầu ngày càng cao trong việc phát triển và duy trì các dịch vụ web phức tạp trong môi trường công nghệ ngày nay.

Chương 3

Phương pháp

Phương pháp đánh giá điểm thưởng được trình bày trong chương này tập trung vào việc tác động đến mức độ phủ của bài toán kiểm thử API, đồng thời đảm bảo tính hiệu quả và toàn diện trong quá trình đánh giá. Để đạt được mục tiêu này, phương pháp đề xuất xét đến nhiều khía cạnh quan trọng khác nhau của API và quá trình kiểm thử. Bên cạnh đó, chương này cũng trình bày về việc mở rộng và tinh chỉnh nguồn sinh dữ liệu mới cho công cụ, dựa trên việc khai thác thông tin thu thập trong quá trình kiểm thử.

3.1 Hướng tiếp cận

3.1.1 Phương pháp đánh giá điểm thưởng trong bài toán kiểm thử API sử dụng học tăng cường

Hiện tại, ARAT-RL đánh giá điểm thưởng cho các hành động dựa trên mã trạng thái trả về. Tuy nhiên, phương pháp này bộc lộ một số điểm yếu, bao gồm khả năng xử lý thông tin trả về hạn chế và hiệu quả tìm lỗi chưa tối ưu. Thứ nhất, phương pháp này không xử lý tốt các thông tin trả về ngoài mã trạng thái, dẫn đến việc bỏ sót những dữ liệu quan trọng có thể ảnh hưởng đến việc đánh giá điểm thưởng. Thứ hai, việc chỉ tập trung khai thác mã trạng thái trả về khiến quá trình tìm lỗi chưa hiệu quả, bỏ sót nhiều lỗi tiềm ẩn gây ra bởi các thành phần khác.

Báo cáo này trình bày phương pháp đánh giá nhằm khắc phục những điểm yếu nêu trên. Phương pháp này hướng đến hai mục tiêu chính: ưu tiên định hướng sinh các ca kiểm thử vào vùng có lỗi (những yêu cầu trả về mã trạng thái `5xx`) và đa dạng hóa phản hồi để tối ưu hóa độ phủ đầu ra. Nhờ đó, về lâu dài, công cụ có khả năng phát hiện lỗi hiệu quả hơn. Để đạt được hai mục tiêu này, phương pháp đánh giá áp dụng đồng thời hai chiến lược chính.

Chiến lược đầu tiên tập trung vào việc ưu tiên định hướng sinh các ca kiểm thử vào vùng có lỗi. Cụ thể, điều này bao gồm việc phân loại các mã trạng thái trả về từ các yêu cầu API, từ đó ưu tiên kiểm thử các thao tác trả về mã trạng thái

`5xx`, tức là những yêu cầu có lỗi xảy ra. Bằng cách tập trung vào các mã trạng thái này, chúng ta có thể nhanh chóng xác định và xử lý các vấn đề nghiêm trọng nhất trong hệ thống.

Ngoài ra, việc xem xét các yêu cầu có mã trả về nằm ngoài đặc tả API cũng rất quan trọng. Các yêu cầu này có thể chỉ ra những lỗi tiềm ẩn không được dự đoán trước, góp phần tăng cường độ phủ kiểm thử và khả năng phát hiện lỗi. Điều này bao gồm cả những mã trạng thái bất thường hoặc không mong đợi, giúp đảm bảo rằng hệ thống hoạt động đúng theo đặc tả trong mọi tình huống.

Việc kết hợp phân loại mã trạng thái và xem xét các yêu cầu nằm ngoài đặc tả API sẽ định hướng việc lựa chọn các nguồn sinh dữ liệu đầu vào một cách hiệu quả. Bằng cách này, chúng ta có thể tối ưu hóa khả năng phát hiện lỗi của công cụ kiểm thử, đảm bảo rằng các nguồn dữ liệu được chọn có khả năng cao phát hiện các vấn đề trong hệ thống. Nhờ đó, quá trình kiểm thử trở nên hiệu quả hơn, giúp phát hiện và sửa chữa lỗi một cách nhanh chóng và chính xác.

Chiến lược thứ hai tập trung vào việc đa dạng hóa phản hồi để tối ưu hóa độ phủ đầu ra. Phương pháp này khuyến khích việc tìm kiếm và phát hiện các lỗi mới thay vì chỉ tập trung vào việc tái tạo những lỗi đã được ghi nhận trước đây. Cách tiếp cận này nhắm đến việc tạo ra các ca kiểm thử mới có khả năng cao phát hiện ra các lỗi tiềm ẩn chưa được nhận diện trước đó. Bằng cách này, chúng ta có thể mở rộng phạm vi kiểm thử và đảm bảo rằng hệ thống được kiểm tra một cách toàn diện và kỹ lưỡng.

Một phần quan trọng của chiến lược này là việc đánh giá, so sánh với danh sách các ca kiểm thử đã được thực hiện trước đó. Thông qua việc phân tích này, giải pháp nhằm phát triển các ca kiểm thử mới và đa dạng hơn, tránh việc lặp lại các ca kiểm thử tương tự, vốn có thể dẫn đến sự trùng lặp và lãng phí tài nguyên. Đồng thời, việc khuyến khích sự đa dạng của các ca kiểm thử không chỉ giúp phát hiện ra các lỗi mới mà còn đảm bảo rằng API được kiểm thử dưới nhiều điều kiện và kịch bản khác nhau. Điều này làm tăng khả năng phát hiện ra các lỗi mà chỉ xuất hiện trong những tình huống đặc biệt hoặc ít phổ biến. Từ đó, công cụ kiểm thử có thể đảm bảo rằng hệ thống hoạt động đúng và ổn định trong mọi tình huống, từ đó cải thiện chất lượng và độ tin cậy của API.

3.1.2 Mở rộng nguồn sinh dữ liệu cho công cụ ARAT-RL

Hiện tại, ARAT-RL hỗ trợ 5 nguồn sinh dữ liệu đầu vào với 5 chiến thuật khác nhau: giá trị ví dụ trong tài liệu, giá trị được xây dựng dựa trên phản hồi trước đó, giá trị được xây dựng dựa trên yêu cầu trước đó, giá trị mặc định, giá trị ngẫu nhiên dựa trên các ràng buộc của tài liệu. Mỗi chiến thuật này đều có đặc điểm riêng biệt và phục vụ cho các mục đích khác nhau trong quá trình sinh dữ liệu.

Qua tìm hiểu mã nguồn, tôi nhận thấy công cụ hỗ trợ mở rộng các nguồn sinh mã nguồn theo chiều ngang, nâng cao tính đa dạng của dữ liệu đầu vào. Điều này mang lại lợi ích to lớn, giúp ARAT-RL có thể thích ứng với nhiều trường hợp sử dụng khác nhau hơn. Việc mở rộng các nguồn sinh dữ liệu không chỉ làm phong phú thêm các loại dữ liệu đầu vào mà còn cải thiện khả năng của hệ thống trong việc xử lý các tình huống phức tạp và đa dạng.

Bên cạnh đó, đối với nguồn sinh dữ liệu ngẫu nhiên, hiện tại ARAT-RL đang sử dụng hàm sinh ngẫu nhiên mặc định của Python. Tuy nhiên, hàm sinh ngẫu nhiên mặc định này có thể không đáp ứng được tất cả các yêu cầu về độ chính xác và hiệu quả. Việc sử dụng các thư viện sinh ngẫu nhiên bên thứ ba có thể mang lại nhiều lợi ích. Các thư viện này cung cấp các thuật toán sinh ngẫu nhiên phức tạp và đa dạng hơn, giúp cải thiện hiệu quả và độ chính xác của việc tạo dữ liệu ngẫu nhiên. Sử dụng các thư viện bên thứ ba cũng có thể giúp tối ưu hóa hiệu suất hệ thống và giảm thời gian xử lý, đặc biệt là khi làm việc với các bộ dữ liệu lớn và phức tạp.

3.2 Thiết kế giải pháp

3.2.1 Phương pháp đánh giá điểm thưởng trong bài toán kiểm thử API sử dụng học tăng cường

Hình **??** trình bày tổng quan về giải pháp đánh giá điểm thưởng. Đầu vào của quá trình này là thông tin về phản hồi nhận từ hệ thống, là kết quả của một yêu cầu. Trong đó, mã trạng thái của phản hồi cho biết hoạt động đó thành công hay không. Mỗi mã trạng thái phản hồi có ý nghĩa riêng, được phân loại thành 4 nhóm, mỗi nhóm có ý nghĩa riêng và đóng góp khác nhau vào quá trình kiểm thử:

- **Không phản hồi (No response):** Đây là trường hợp không mong muốn của

một lời gọi API. Việc không nhận được phản hồi có nghĩa là không có thông tin để xác định liệu SUT (hệ thống đang được kiểm thử) đang chạy chính xác hay gặp vấn đề với môi trường mạng.

- **401 (Chưa được chứng thực) và 403 (Không được phép truy cập):** Các mã trạng thái HTTP 401 (Chưa được chứng thực) và 403 (Không được phép truy cập) chỉ ra các vấn đề liên quan đến quyền truy cập của người dùng đối với các tài nguyên API. Khi gặp phải các mã trạng thái này trong quá trình kiểm thử, ta cần phải lưu ý một số điểm. Những điểm này sẽ giúp hiểu rõ hơn về nguyên nhân và cách xử lý tình huống.

  Mã trạng thái 401 (Chưa được chứng thực) được trả về khi người dùng hoặc hệ thống gửi yêu cầu tới máy chủ mà không kèm theo thông tin xác thực hợp lệ. Điều này có nghĩa là máy chủ yêu cầu người dùng cung cấp thông tin xác thực (như tên đăng nhập và mật khẩu) nhưng thông tin này hoặc không được cung cấp hoặc không chính xác. Khi nhận mã 401, máy chủ không thực hiện bất kỳ hành động nào đối với tài nguyên yêu cầu vì không có thông tin xác thực. Do đó, mã trạng thái này không có giá trị trong quá trình kiểm thử tính năng của SUT (System Under Test) vì không thể tiến hành bất kỳ kiểm tra nào liên quan đến hành động cụ thể trên tài nguyên API.

  Mã trạng thái 403 (Không được phép truy cập) được trả về khi người dùng hoặc hệ thống đã xác thực thành công nhưng không có quyền truy cập vào tài nguyên được yêu cầu. Điều này xảy ra khi người dùng không có quyền hoặc vai trò cần thiết để thực hiện hành động mong muốn trên tài nguyên đó. Khi nhận mã 403, máy chủ từ chối thực hiện hành động yêu cầu vì vấn đề quyền truy cập. Điều này cũng có nghĩa là không có bất kỳ thay đổi nào xảy ra trên tài nguyên API và không có thông tin hữu ích nào để khai thác cho mục đích kiểm thử.

  Tóm lại, cả hai mã trạng thái này đều liên quan đến vấn đề quyền truy cập và không cung cấp thông tin hữu ích cho quá trình kiểm thử tính năng của SUT. Chúng chỉ ra rằng vấn đề không nằm ở SUT mà ở việc cấp quyền truy cập. Vì vậy, khi gặp phải các mã này trong quá trình kiểm thử, cần kiểm tra lại thông tin xác thực và quyền truy cập của người dùng trước khi tiếp tục kiểm tra các tính năng của hệ thống.

- **Các nhóm 4xx, 3xx và 2xx:** Các nhóm này thể hiện những trạng thái của

SUT. Cụ thể, nhóm `2xx` và `3xx` biểu thị yêu cầu đã được thực hiện thành công. Ngược lại, nhóm `4xx` biểu thị yêu cầu thực hiện thất bại, xuất hiện lỗi từ phía máy khách. Mặc dù các nhóm mã trạng thái này không trực tiếp giúp tìm kiếm lỗi bên phía máy chủ nhưng chúng cũng giúp ta mở rộng tập các mã trạng thái được kiểm thử, tăng độ phủ đầu ra.

- **Nhóm 5xx:** Mã trạng thái `500` cho biết có một số lỗi trạng thái nội bộ của SUT trong khi xử lý đầu vào yêu cầu tương ứng. Đây là kết quả không mong muốn nhất từ một lời gọi API, nhưng mặt khác lại là phản hồi có giá trị nhất trong kiểm thử API.

Dựa trên hướng tiếp cận cốt lõi, giải pháp thiết kế cơ chế điểm thưởng hoạt động theo hai vòng riêng biệt. Mỗi vòng đánh giá bốn nhóm mã trạng thái được đề cập ở trên từ các góc độ khác nhau. Vòng một nhằm mục đích đánh giá điểm thưởng phù hợp dựa trên mã trạng thái trả về, đánh giá khả năng phát hiện lỗi của lời gọi API vừa được thực hiện. Điều này phù hợp với mục tiêu "Ưu tiên định hướng sinh các ca kiểm thử vào vùng có lỗi". Vòng thứ hai tính toán điểm thưởng dựa trên tính mới của phản hồi. Chi tiết của từng vòng được trình bày bên dưới.

Vòng 1: Đánh giá điểm thưởng Dựa trên Mã Trạng thái

Ở vòng 1, điểm thưởng sẽ được đánh giá dựa trên mã trạng thái trả vè, đánh giá dựa trên khả năng phát hiện lỗi của yêu cầu vừa được thực hiện. Dựa trên 4 nhóm mã trạng thái đã được phân loại, phương pháp sẽ sẽ đưa ra điểm thưởng phù hợp dựa trên các lý luận sau đây:

- **Không phản hồi:** Sự thiếu phản hồi có thể đến từ một số yếu tố: thứ nhất, máy chủ quá tải dẫn đến hết thời gian chờ; thứ hai, xử lý mã nguồn chậm dẫn đến hết thời gian chờ; và thứ ba, sự hiện diện của lỗi khiến việc trả về phản hồi bị chặn. Đối với lý do thứ nhất, công cụ nên tránh tiếp tục kiểm tra điểm truy cập đó, vì vậy điểm thưởng sẽ mang tính tiêu cực. Tuy nhiên, tình trạng quá tải có thể chỉ là tạm thời, do đó có thể cho phép kiểm thử lại ở giai đoạn sau của quá trình kiểm thử. Mặc dù điểm thưởng trong trường hợp này mang tính tiêu cực, nhưng chúng không nghiêm trọng bằng trường hợp lỗi 401 và 403. Đối với lý do thứ hai và thứ ba, đây là những trường hợp lỗi. Dù mang lại giá trị cho mục đích kiểm thử, nhưng việc thiếu phản hồi sẽ không mang

lại giá trị về mặt khai thác thêm thông tin và mô hình không thể tiếp nhận kiến thức từ các yêu cầu này. Giải pháp chung cho các trường hợp không có phản hồi là tránh tiếp tục kiểm tra điểm truy cập đó.

- **401 (Không được xác thực), 403 (Không được ủy quyền):** Các điểm truy cập (endpoint) không được xác thực thường trả về mã 401 hoặc 403. Nếu không thể gửi yêu cầu đến điểm truy cập do xác thực, thì điểm truy cập đó sẽ không bao giờ nên được kiểm tra trong quá trình kiểm thử. Chúng ta nên tránh kiểm tra thêm và không kiểm tra lại bằng việc giảm đáng kể điểm thưởng.

- **Các nhóm 4xx, 3xx và 2xx:** Như đã đề cập ở trên, các nhóm này giúp tăng độ phủ đầu ra. Tuy nhiên, những phát hiện này không đóng góp đáng kể vào việc phát hiện lỗi. Do đó, với các nhóm `2xx` và `3xx`, điểm thưởng cho các trường hợp này vẫn mang tính tích cực, nhưng ở mức độ vừa phải. Tương tự, với nhóm `4xx`, điểm thưởng mang tính tiêu cực, nhưng cũng ở mức độ vừa phải. Dù vậy, nếu mã trạng thái trả về không được định nghĩa trong đặc tả API, điểm thưởng sẽ được điều chỉnh tăng thêm để khuyến khích việc phủ những mã trạng thái mới. Việc phát hiện mã trạng thái ngoài đặc tả cũng được coi là khám phá mới nhưng nó không được coi là quan trọng, do đó chỉ nhận được điểm thưởng vừa phải.

- **Nhóm 5xx:** Trong kiểm thử API, nhóm mã trạng thái này luôn là mục tiêu chính của kiểm thử API và điểm thưởng của chúng sẽ theo hướng tích cực. Đặc biệt, trong nhóm này, lỗi `500` có ý nghĩa quan trọng. Lỗi `500`, hay "Internal Server Error" thường chỉ ra rằng có sự cố bên trong máy chủ, gây ra bởi một lỗi trong mã nguồn, lỗi cấu hình, hoặc một tình huống không lường trước được trong quá trình xử lý yêu cầu. Vậy nên, việc phát hiện và xác định lỗi `500` được ưu tiên hơn cả, mang ý nghĩa là tìm thấy lỗi trực tiếp của hệ thống. Điều này cực kỳ có giá trị trong quá trình kiểm thử vì nó giúp phát hiện ra các lỗ hổng và vấn đề tiềm ẩn mà hệ thống cần khắc phục.

Vòng 2: Đánh giá Thưởng Dựa trên Tính Mới của Phản Hồi

Ở vòng 2, điểm thưởng được đánh giá dựa trên mức độ mới của phản hồi. Một phản hồi được coi là mới khi xuất hiện ít thường xuyên hoặc chưa từng xuất hiện trước đây, với mục tiêu là tăng phạm vi kiểm thử. Ngược lại, một phản hồi được

coi là quen thuộc khi nó được trả về thường xuyên. Ngay cả với một phản hồi mang mã trạng thái `5xx`, nó không mang lại nhiều ý nghĩa nếu xuất hiện quá nhiều lần. Việc kiểm tra tính mới cũ của phản hồi giúp tránh tạo ra các yêu cầu tương tự lặp đi lặp lại, từ đó tăng được hiệu quả trong việc kiểm thử.

Khi cài đặt thuật toán, điểm thưởng đạt được ở vòng 1 sẽ đóng vai trò là đầu vào cho vòng 2. Ở vòng 2, một hệ số, được ký hiệu là d, được sử dụng để làm tham chiếu đánh giá tính mới của phản hồi. Điều này được thực hiện bằng cách nhân điểm thu được ở vòng 1 với hệ số d (phương trình 3.1).

$$reward_{r2} = reward_{r1} \times d \tag{3.1}$$

Giải pháp tổng quát cho vòng 2 là so sánh phản hồi hiện tại với k phản hồi gần nhất trong lịch sử trên cùng một hoạt động (operation) và cùng một mã trạng thái để đánh giá tính mới của phản hồi hiện tại. Phương pháp này xác định số lượng phản hồi trước đây tương đồng với phản hồi hiện tại bằng cách so sánh với k phản hồi gần nhất. Hệ số d có xu hướng giảm dần nếu số lượng phản hồi tương đồng tăng lên và ngược lại. Giá trị cụ thể hệ số d cũng phụ thuộc vào từng loại nhóm mã trạng thái đã trình bày ở vòng 1. Mục đích chung của vòng này là tránh lặp lại cùng một phản hồi một cách thường xuyên.

Tiêu chí đánh giá sự khác biệt giữa hai phản hồi cần được xem xét một cách chi tiết và hệ thống. Đầu tiên, khi nội dung phản hồi là stack trace của lỗi `5xx`, nội dung stack trace sẽ được so sánh để xác định sự tương đồng. Hai stack trace được coi là giống nhau nếu chúng hoàn toàn trùng khớp về nội dung. Đối với phản hồi dưới dạng HTML, việc so sánh cấu trúc cây DOM của hai trang HTML là cần thiết. Hai trang HTML được coi là khác nhau nếu cấu trúc của hai cây DOM khác biệt. Tương tự, khi nội dung phản hồi có dạng JSON, cấu trúc của hai đối tượng JSON sẽ được so sánh. Hai đối tượng JSON được coi là khác nhau nếu có sự khác biệt về độ sâu, số lượng trường dữ liệu, hoặc tên các khóa (key) trong các trường dữ liệu. Cuối cùng, đối với nội dung phản hồi có dạng chuỗi ký tự, sự khác biệt về số từ trong chuỗi sẽ được coi là tiêu chí để đánh giá hai chuỗi khác nhau. Mỗi phương pháp phân loại này đảm bảo rằng các tiêu chí đánh giá sự khác biệt giữa hai phản hồi đều được thực hiện một cách chính xác và toàn diện, giúp cải thiện độ chính xác trong quá trình phân tích và xử lý dữ liệu.

3.2.2 Mở rộng và tinh chỉnh nguồn sinh dữ liệu cho công cụ ARAT-RL

Sửa đổi nguồn sinh dữ liệu ngẫu nhiên

Hiện tại, công cụ ARAT-RL sử dụng thư viện tích hợp của Python để sinh dữ liệu ngẫu nhiên. Mặc dù thư viện này tiện dụng, nhưng nó bộc lộ một số hạn chế. Đầu tiên, thư viện sử dụng các thuật toán PRNGs (Pseudo Random Number Generators) để tạo các số ngẫu nhiên. Điều này có nghĩa là các số được tạo ra có vẻ ngẫu nhiên nhưng thực chất dựa trên giá trị hạt giống (seed) ban đầu, do đó không thực sự ngẫu nhiên. Thêm vào đó, một số hàm trong thư viện có thể tạo ra các số không phân bố đều trên toàn phạm vi, dẫn đến phân bố không đồng đều. Cuối cùng, thuật toán PRNGs có khả năng lặp lại cao vì bị ảnh hưởng bởi giá trị hạt giống ban đầu, có thể sinh ra chuỗi số giống nhau, làm giảm tính ngẫu nhiên của dữ liệu.

Để giải quyết những hạn chế này, giải pháp đề xuất là thay thế thư viện hiện tại bằng việc sử dụng thư viện bên thứ ba. Giải pháp lựa chọn sử dụng Hypothesis, một thư viện kiểm thử mờ mạnh mẽ có chức năng sinh dữ liệu cho các ca kiểm thử [6]. Việc sử dụng Hypothesis không chỉ giúp cải thiện tính ngẫu nhiên thực sự của dữ liệu mà còn được kỳ vọng tăng độ phủ trong quá trình kiểm thử. Trong bối cảnh các yêu cầu kiểm thử ngày càng phức tạp và đa dạng, việc áp dụng các công cụ mạnh mẽ như Hypothesis là cần thiết để đảm bảo chất lượng và hiệu quả của quá trình kiểm thử. Hypothesis không chỉ giải quyết các vấn đề hiện tại mà còn chuẩn bị cho ARAT-RL khả năng ứng phó với các thách thức kiểm thử trong tương lai.

Thêm nguồn sinh dữ liệu dựa trên việc biến đổi các giá trị thành công trước đó

Những yêu cầu được thực hiện thành công luôn mang ý nghĩa quan trọng trong kiểm thử. Khi một yêu cầu được thực hiện thành công, điều này cho thấy dữ liệu được gửi đi trong yêu cầu đó hoạt động chính xác và tuân thủ các quy tắc của hệ thống. Lưu trữ và biến đổi dữ liệu này không chỉ tăng tốc độ phát hiện lỗi mà còn có khả năng tìm ra các lỗi mới. Dựa trên ý tưởng này, giải pháp đã xây dựng một nguồn sinh dữ liệu mới, biến đổi dữ liệu từ các yêu cầu thành công trước đây.

Việc biến đổi dữ liệu ở đây nhằm mục đích sinh dữ liệu không tuân theo đặc tả. Giải pháp được đề xuất gồm hai phương pháp: tạo dữ liệu không đúng với mô tả của các trường dữ liệu trong đặc tả hoặc loại bỏ một số trường dữ liệu bắt buộc trong lược đồ tham số. Phương pháp được sử dụng sẽ được lựa chọn một cách ngẫu nhiên.

Với phương pháp tạo dữ liệu không tuân theo mô tả, giải pháp thực hiện duyệt qua từng trường dữ liệu trong lược đồ. Việc quyết định thực hiện biến đổi trường dữ liệu được phụ thuộc vào một số ngẫu nhiên. Nếu số ngẫu nhiên lớn hơn hằng số $mutation\_rate$ đã được định nghĩa., giải pháp sẽ tiến hành biến đổi. Giải pháp ở đây trình bày hai cách tạo dữ liệu không tuân theo đặc tả: thay đổi kiểu dữ liệu của trường dữ liệu mục tiêu hoặc tạo dữ liệu vượt quá các ràng buộc của trường dữ liệu. Việc đổi kiểu dữ liệu giúp kiểm tra tính linh hoạt và khả năng xử lý của hệ thống đối với các kiểu dữ liệu khác nhau. Ngoài ra, việc sinh dữ liệu ngẫu nhiên nằm ngoài ràng buộc của trường dữ liệu cũng là một chiến lược quan trọng. Dữ liệu này có thể không tuân thủ hoàn toàn các quy tắc đã được định nghĩa trước, nhằm mục đích kiểm tra xem cách hệ thống phản ứng với các giá trị bất ngờ hoặc không hợp lệ.

Với phương pháp loại bỏ một số trường dữ liệu, giải pháp loại bỏ ngẫu nhiên một số các trường dữ liệu cần có được mô tả trong lược đồ. Bằng cách loại bỏ các trường dữ liệu, ta có thể mô phỏng các tình huống thực tế trong đó dữ liệu có thể bị thiếu sót hoặc không đầy đủ do nhiều nguyên nhân khác nhau như lỗi nhập liệu, truyền tải dữ liệu bị gián đoạn, hoặc nguồn dữ liệu không chính xác. Những biến đổi này giúp tạo ra các kịch bản kiểm thử phức tạp hơn, đảm bảo rằng hệ thống có thể xử lý nhiều loại tình huống khác nhau.

Chương 4

Thực nghiệm và đánh giá

Chương này sẽ tập trung trình bày về quá trình thực nghiệm và đánh giá phương pháp đánh giá điểm thưởng dựa trên thiết kế đã mô tả chi tiết trong phần trước. Phần thực nghiệm sẽ đi sâu vào việc áp dụng phương pháp để cải thiện hiệu quả công cụ ARAT-RL và so sánh hiệu suất của phương pháp mới với phương pháp đánh giá điểm thưởng mặc định của ARAT-RL. Dựa trên kết quả thu được, tôi sẽ đưa ra các nhận xét và kết luận về giải pháp đề xuất.

4.1 Dữ liệu

Bảng **??** mô tả chi tiết về các hệ thống thực nghiệm. Bộ dữ liệu sử dụng trong báo cáo này bao gồm 10 hệ thống kiểm thử được thực nghiệm trong bài báo ARAT-RL. Bộ dữ liệu này chứa khoảng 860.000 dòng mã nguồn và 197 hoạt động (operation), được trích xuất từ một bài báo gần đây với mục đích xây dựng bộ dữ liệu cho kiểm thử API [12]. Theo Myeongsoo Kim và các cộng sự, những hệ thống này đảm bảo các điều kiện cần thiết để thực hiện kiểm thử hiệu quả. Cụ thể, chúng cung cấp đặc tả OpenAPI, có thể biên dịch và hoạt động tốt, và không phụ thuộc vào các dịch vụ bên ngoài có giới hạn yêu cầu. Điều này đảm bảo hiệu suất của công cụ kiểm thử không bị ảnh hưởng bởi các yếu tố ngoại vi. Thông tin này giúp đảm bảo tính nhất quán và độ tin cậy của các kết quả được thu thập và phân tích trong nghiên cứu.

4.2 Quy trình thực nghiệm

Quy trình thực nghiệm bắt đầu bằng việc xác định mục tiêu chính là cải thiện hiệu suất phát hiện lỗi của công cụ. Mục tiêu này rất quan trọng vì nó xem xét khả năng, tốc độ của công cụ trong môi trường thực tế. Để đạt được mục tiêu này, ba thực nghiệm đã được tiến hành, mỗi thực nghiệm được thiết kế để kiểm tra một khía cạnh cụ thể của hiệu suất công cụ.

Thực nghiệm đầu tiên tập trung vào việc đánh giá số lượng lỗi mà công cụ phát hiện được sau khi đã được cải tiến so với phiên bản gốc. Quy trình này bao gồm việc chạy công cụ trên một tập dữ liệu thử nghiệm, sau đó so sánh kết quả thu được với phiên bản cũ. Bằng cách này, thực nghiệm có thể đánh giá sự cải thiện trong khả năng phát hiện lỗi của công cụ sau khi thực hiện các điều chỉnh và cải tiến.

Thực nghiệm thứ hai tập trung vào so sánh tốc độ tìm lỗi giữa phiên bản đã được cải tiến và công cụ gốc. Để làm được điều này, thực nghiệm đã lập biểu đồ thể hiện sự tương quan giữa số lỗi tìm được và số yêu cầu gửi đi giữa hai phiên bản. Qua đó, ta có thể xác định ảnh hưởng việc cải tiến đến tốc độ phát hiện lỗi.

Thực nghiệm thứ ba đánh giá ảnh hưởng của cải tiến nguồn sinh dữ liệu đến tốc độ tìm lỗi trong công cụ áp dụng phương pháp đánh giá điểm thưởng mới. Tương tự với thực nghiệm thứ hai, biểu đồ thể hiện sự tương quan giữa số lỗi tìm được và số yêu cầu gửi đi giữa hai phiên bản đã được lập. Điều này giúp xác định liệu việc cải tiến về nguồn sinh dữ liệu đầu vào có ảnh hưởng tích cực đến tốc độ phát hiện lỗi hay không.

Tổng hợp kết quả từ cả ba thực nghiệm sẽ cung cấp cái nhìn toàn diện về hiệu suất của công cụ phát hiện lỗi sau khi được cải tiến. Thông tin thu được từ các thực nghiệm này sẽ là cơ sở quan trọng để đánh giá và điều chỉnh chiến lược phát triển tiếp theo của dự án, đảm bảo rằng công cụ ngày càng hoàn thiện và hiệu quả hơn trong việc phát hiện và xử lý lỗi.

4.3 Độ đo đánh giá

Tương tự như ARAT-RL, hiệu quả của phương pháp đề xuất được đánh giá thông qua thước đo hiệu suất tìm lỗi, tập trung vào việc xác định các trường hợp duy nhất của mã lỗi `500`, biểu thị sự cố phía máy chủ. Nếu hệ thống cung cấp stack trace khi gặp phản hồi dạng `500`, công cụ thu thập stack trace và tiến hành phân tích, so sánh. Mỗi lỗi bên phía hệ thống được định nghĩa là phản hồi dạng 500 có stack trace riêng biệt. Trong phần lớn các trường hợp, các lỗi tìm được thuộc thể loại này. Ngoại lệ, nếu hệ thống không hỗ trợ trả về stack trace, công cụ sẽ phân tích nội dung phản hồi. Sau khi loại bỏ các thành phần không liên quan như mốc thời gian, giải pháp phân loại các trường hợp phản hồi dạng `500` có nội dung độc

nhất thành các lỗi riêng lẻ.

Với từng thực nghiệm, hiệu năng sẽ được đánh giá trên từng thang đo khác nhau. Trong thực nghiệm đầu tiên, vì tập trung đánh giá số lượng lỗi mà công cụ phát hiện được, thang đo đánh giá được xác định là trung bình số các lỗi duy nhất được xác định trong phiên thực nghiệm kéo dài một giờ. Để giảm thiểu ảnh hưởng của tính ngẫu nhiên, mỗi thực nghiệm được lặp lại 10 lần và tính trung bình số các lỗi được tìm thấy trong tất cả các thực nghiệm. Thang đo này tương tự với phương pháp mà ARAT-RL áp dụng.

Trong hai thực nghiệm tiếp theo, giải pháp tập trung vào đánh giá tốc độ tìm lỗi, thang đo đánh giá được xác định là số lượng lỗi tìm được các một mốc yêu cầu (request). Tương tự, để giảm thiểu ảnh hưởng của tính ngẫu nhiên, mỗi thực nghiệm được lặp lại 10 lần. Việc này đảm bảo rằng các kết quả thu được có độ tin cậy cao và phản ánh chính xác hiệu suất của công cụ sau khi áp dụng các cải tiến.

4.4 Kết quả thực nghiệm

4.4.1 Đánh giá số lượng lỗi mà công cụ phát hiện được sau khi đã được cải tiến so với phiên bản gốc

Bảng **??** trình bày tổng số lỗi được phát hiện bởi từng công cụ trong 10 lần chạy trên tập các hệ thống thực nghiệm. Lưu ý rằng tổng số lỗi này có thể bao gồm việc phát hiện nhiều lần cùng một lỗi trong các lần chạy khác nhau. Dựa theo bảng **??**, công cụ sau khi cải tiến cho thấy khả năng phát hiện lỗi hiệu quả, vượt trội hoặc sánh ngang với công cụ gốc trong mọi hệ thống kiểm thử và xác định trung bình 126,9 lỗi trên các dịch vụ. Khả năng phát hiện lỗi của công cụ cải tiến đặc biệt rõ ràng trong các dịch vụ "Language Tool" và "Person Controller", lần lượt xác định trung bình 13,7 lỗi và 107,2 lỗi. Điều này được giải thích là các hệ thống này có bộ tham số lớn hơn, cho phép kiểm thử với nhiều tổ hợp tham số đa dạng, dẫn đến độ phủ đầu ra lớn hơn và xác suất phát hiện lỗi cao hơn.

4.4.2 So sánh về tốc độ tìm lỗi giữa phiên bản cải tiến và công cụ gốc

Hình **??** minh họa tổng quan tốc độ tìm lỗi giữa phiên bản cải tiến và phiên bản gốc, cung cấp cái nhìn chi tiết hơn về quá trình kiểm thử của công cụ. Trục hoành

đại diện cho số lượng yêu cầu đã được thực hiện, trong khi trục tung biểu thị số lượng lỗi tìm được. Tổng quan, công cụ sau khi cải tiến cho thấy hiệu quả vượt trội rõ rệt so với phiên bản gốc. Ta có thể nhận xét đồ thị này dựa trên ba khía cạnh: số lượng lỗi tìm được trên mỗi mốc yêu cầu, thời điểm bão hòa và số lượng yêu cầu được sinh ra.

Với tiêu chí đầu tiên, mặc dù có sự thua thiệt ở các mốc yêu cầu đầu tiên, nhìn tổng thể, giải pháp đánh giá điểm thưởng mới đã mang lại hiệu quả vượt trội so với công cụ gốc. Ta có thể thấy, khi thời gian chạy tăng, phiên bản cải tiến càng bỏ xa phiên bản gốc về số lượng lỗi tìm được trên mỗi mốc yêu cầu. Tại mốc yêu cầu cuối cùng mà phiên bản gốc có thể thực hiện, phiên bản cải tiến tìm được nhiều hơn gần 15 lỗi. Điều này cho thấy phiên bản cải tiến có khả năng tìm kiếm lỗi hiệu quả hơn, cần ít yêu cầu hơn để đạt được số lượng lỗi nhất định. Sự thua thiệt ban đầu ở các mốc yêu cầu đầu tiên có thể lý giải rằng, do phương pháp mới xét đến nhiều yếu tố hơn trong quá trình đánh giá, không tập trung một cách trực tiếp vào việc tìm lỗi, làm giảm tốc độ tìm lỗi ban đầu của công cụ.

Với tiêu chí thứ hai, điểm bão hòa của công cụ gốc xuất hiện sớm hơn so với công cụ sau khi cải tiến. Ta có thể thấy rằng, vào khoảng yêu cầu thứ 35.000, công cụ gốc đã đạt điểm bão hòa, đồ thị đi theo hướng nằm ngang và số lượng lỗi tìm được không tăng. Trong khi đó, phiên bản cải tiến tiếp tục cho thấy khả năng tìm ra lỗi. Điều này cho thấy phương pháp đánh giá điểm thưởng mới có chiều sâu và tác động xuyên suốt quá trình kiểm thử.

Với tiêu chí thứ ba, ta có thể thấy công cụ sinh được nhiều yêu cầu hơn trong cùng 1 phiên thực nghiệm. Đồ thị trên được thực nghiệm trên 10 phiên để loại bỏ tính ngẫu nhiên trong quá trình thực nghiệm. Một trong những lý do có thể được đưa ra trong quá trình này, đó là việc mở rộng và tinh chỉnh các nguồn sinh dữ liệu, đã có ảnh hưởng đến tốc độ sinh yêu cầu của công cụ. Cụ thể hơn, đó là việc áp dụng thư viện Hypothesis vào quá trình sinh dữ liệu.

4.4.3 Đánh giá ảnh hưởng của cải tiến nguồn sinh dữ liệu đến tốc độ tìm lỗi trong công cụ áp dụng phương pháp đánh giá điểm thưởng mới

Để đánh giá ảnh hưởng của cải tiến nguồn sinh dữ liệu đến tốc độ tìm lỗi trong công cụ áp dụng phương pháp đánh giá điểm thưởng mới, ta làm một thử nghiệm tương tự với thử nghiệm số hai. Với các tiêu chí đánh giá tương tự thử nghiệm trước, ở đây, ta có thể thấy sự khác biệt trên hai khía cạnh sau đây: số lượng lỗi tìm được trên mỗi mốc yêu cầu và số lượng yêu cầu sinh ra

Về tiêu chí đầu tiên, hình **??** cho thấy tốc độ tìm lỗi tổng thể được cải thiện khi áp dụng mở rộng và tinh chỉnh các nguồn sinh dữ liệu.Đặc biệt trong giai đoạn đầu (khoảng 25.000 yêu cầu đầu tiên), sự chênh lệch về hiệu quả tìm lỗi giữa hai phiên bản được thể hiện rõ ràng nhất. Sự khác biệt lên đến 10 lỗi đã được ghi nhận tại mốc 13.000 yêu cầu, cho thấy phiên bản cải tiến có khả năng phát hiện lỗi vượt trội so với phiên bản gốc trong giai đoạn này.

Về tiêu chí thứ hai, như đã giải thích ở trên, việc tích hợp thư viện Hypothesis trong quá trình mở rộng và tinh chỉnh nguồn sinh dữ liệu đầu vào đã giúp nâng cao khả năng sinh yêu cầu trong cùng một phiên thực nghiệm. Cụ thể, phiên bản cải tiến đã sinh thêm 18.000 yêu cầu, tương đương tăng 25% so với phiên bản gốc. Khả năng sinh nhiều yêu cầu hơn giúp phiên bản cải tiến khám phá nhiều trường hợp kiểm thử hơn, nâng cao tiềm năng tìm ra nhiều lỗi tiềm ẩn. Ví dụ, phiên bản cải tiến đã tìm thêm lỗi ở yêu cầu thứ 88.500. Mặc dù phiên bản cải tiến chưa cho thấy dấu hiệu bão hòa khi kết thúc phiên kiểm thử, nhưng để đảm bảo tính công bằng, phiên kiểm thử đã được dừng lại.

Kết luận

Kiểm thử tự động API là một chủ đề nhận được sự quan tâm đặc biệt trong lĩnh vực nghiên cứu hiện nay, bởi vì API đóng vai trò quan trọng trong việc kết nối các hệ thống và ứng dụng. Với sự phát triển nhanh chóng của các công nghệ và nhu cầu tích hợp liên tục, việc đảm bảo chất lượng và hiệu suất của các API trở nên cấp thiết. Kiểm thử tự động API không chỉ giúp phát hiện và sửa chữa lỗi nhanh chóng mà còn nâng cao hiệu quả và độ tin cậy của các hệ thống phần mềm.

Khóa luận này đã trình bày một phương pháp đánh giá điểm thưởng tập trung vào việc tối ưu hóa mức độ phủ của bài toán kiểm thử API, đồng thời đảm bảo tính hiệu quả và toàn diện trong quá trình đánh giá. Phương pháp đề xuất đã xét đến nhiều khía cạnh quan trọng của API và quá trình kiểm thử, đồng thời mở rộng và tinh chỉnh nguồn sinh dữ liệu mới cho công cụ, dựa trên việc khai thác thông tin thu thập được trong quá trình kiểm thử.

Qua quá trình thực nghiệm trên một bộ dịch vụ kiểm thử uy tín, kết quả đạt được cho thấy phương pháp đề xuất có hiệu suất tốt hơn so với phiên bản gốc, nâng cao hiệu quả kiểm thử của công cụ. Cụ thể, công cụ được phát triển có thể xác định được 126,9 lỗi riêng biệt với mã trả về dạng `500`, vượt trội hơn 14 lỗi so với ARAT-RL. Về tốc độ, tại đa số thời điểm, công cụ được trình bày đều cho thấy hiệu suất tốt hơn so ARAT-RL. Đặc biệt, tại một số thời điểm, trên cùng một mốc yêu cầu, công cụ này vượt trội hơn với 15 lỗi tìm được nhiều hơn. Những kết quả này chứng minh rằng công cụ không chỉ cải thiện về mặt số lượng lỗi phát hiện được mà còn tăng tốc độ phát hiện lỗi, giúp tiết kiệm tài nguyên và nâng cao chất lượng phần mềm.

Trong tương lai, hướng phát triển của khóa luận này sẽ tập trung vào một số hướng đi quan trọng nhằm tiếp tục nâng cao hiệu quả và khả năng áp dụng của công cụ kiểm thử tự động API. Trước hết, cần tinh chỉnh và tối ưu hóa công cụ bằng cách cải thiện các thuật toán học tăng cường, tích hợp các thuật toán tiên tiến hơn để tăng cường khả năng học hỏi và thích ứng của công cụ trong các tình huống kiểm thử khác nhau. Đặc biệt, sử dụng Deep Q-learning, một phương pháp mạnh mẽ trong học tăng cường sâu, có thể mang lại hiệu quả cao. Deep Q-learning kết hợp giữa Q-learning và mạng nơ-ron sâu (DNN) để tạo ra một công cụ có khả năng xử

lý các trạng thái phức tạp và không gian hành động lớn hơn. Mở rộng phạm vi áp dụng cũng là một mục tiêu quan trọng, với việc hỗ trợ nhiều loại API khác nhau, bao gồm RESTful API, GraphQL, và WebSocket, nhằm đảm bảo tính đa dụng của công cụ trong nhiều bối cảnh ứng dụng khác nhau, đồng thời tích hợp công cụ với các nền tảng và công cụ kiểm thử khác để tạo ra một môi trường kiểm thử toàn diện, phù hợp với nhiều loại dự án và yêu cầu khác nhau[13**?**].

Việc kết hợp với các phương pháp tiên tiến như học máy và phân tích dữ liệu lớn sẽ giúp tự động nhận diện các mẫu lỗi phổ biến và dự đoán các vùng có khả năng xuất hiện lỗi cao trong API. Tích hợp công cụ kiểm thử tự động API vào các quy trình DevOps để thực hiện kiểm thử liên tục (CI/CD) sẽ đảm bảo rằng các lỗi được phát hiện và khắc phục kịp thời trong suốt quá trình phát triển phần mềm. Tăng cường khả năng mở rộng và tùy chỉnh của công cụ cũng là một bước đi cần thiết, bao gồm phát triển một giao diện người dùng trực quan và thân thiện, giúp người dùng dễ dàng cấu hình và sử dụng công cụ mà không cần nhiều kiến thức chuyên môn về kiểm thử, và kết nối công cụ với các hệ thống quản lý lỗi phổ biến để tự động báo cáo và theo dõi lỗi, giúp đội ngũ phát triển phần mềm dễ dàng quản lý và giải quyết các vấn đề phát sinh. Những cải tiến và phát triển này sẽ không chỉ nâng cao khả năng kiểm thử tự động API mà còn đóng góp vào sự phát triển của công nghệ kiểm thử phần mềm, đảm bảo rằng các hệ thống API hoạt động ổn định và đáng tin cậy, đáp ứng được các yêu cầu ngày càng cao của thị trường và người dùng.

Tài liệu tham khảo

[1] Andrea Arcuri. *Many Independent Objective (MIO) Algorithm for Test Suite Generation*, page 3–17. Springer International Publishing, 2017.

[2] Andrea Arcuri. Restful api automated test case generation with evomaster. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 28(1):1–37, 2019.

[3] Vaggelis Atlidakis, Patrice Godefroid, and Marina Polishchuk. Restler: Stateful rest api fuzzing. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 748–758. IEEE, 2019.

[4] Matthias Biehl. *RESTful Api Design*, volume 3. API-University Press, 2016.

[5] Adeel Ehsan, Mohammed Ahmad M. E. Abuhaliqa, Cagatay Catal, and Deepti Mishra. Restful api testing methodologies: Rationale, challenges, and solution directions. *Applied Sciences*, 12(9), 2022.

[6] Paul Falcke, J. Spencer Dodds, Ben Procter, and John Carter. Hypothesis: A library for property-based testing. *ACM SIGPLAN Notices*, 51(1):467–475, 2016.

[7] Roy T. Fielding. Architectural styles and the design of network-based software architectures. *University of California, Irvine*, 7, 2000.

[8] XML-RPC Working Group. Xml-rpc specification, 2000.

[9] Isha, Abhinav Sharma, and M. Revathi. Automated api testing. In *2018 3rd International Conference on Inventive Computation Technologies (ICICT)*, pages 788–791, 2018.

[10] Daniel Jacobson, Greg Brail, and Dan Woods. *APIs: A strategy guide.* " O'Reilly Media, Inc.", 2012.

[11] Myeongsoo Kim, Saurabh Sinha, and Alessandro Orso. Adaptive rest api testing with reinforcement learning. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 446–458. IEEE, 2023.

[12] Myeongsoo Kim, Qi Xin, Saurabh Sinha, and Alessandro Orso. 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*, ISSTA 2022, page 289–301, New York, NY, USA, 2022. Association for Computing Machinery.

[13] Byung Hoon Lee. Graphql, 2012.

[14] Hongliang Liang, Xiaoxiao Pei, Xiaodong Jia, Wuwei Shen, and Jian Zhang. Fuzzing: State of the art. *IEEE Transactions on Reliability*, 67(3):1199–1218, 2018.

[15] Yi Liu, Yuekang Li, Gelei Deng, Yang Liu, Ruiyuan Wan, Runchao Wu, Dandan Ji, Shiheng Xu, and Minli Bao. Morest: model-based restful api testing with execution feedback. In *Proceedings of the 44th International Conference on Software Engineering*, pages 1406–1417, 2022.

[16] Alberto Martin-Lopez, Andrea Arcuri, Sergio Segura, and Antonio Ruiz-Cortés. Black-box and white-box test case generation for restful apis: Enemies or allies? In *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*, pages 231–241, 2021.

[17] Richa Maurya, Keerthi Anil Nambiar, Poornima Babbe, Jyoti Popat Kalokhe, YS Ingle, and NF Shaikh. Application of restful apis in iot: A review. *Int. J. Res. Appl. Sci. Eng. Technol*, 9:145–151, 2021.

[18] Andy Neumann, Nuno Laranjeiro, and Jorge Bernardino. An analysis of public rest web service apis. *IEEE Transactions on Services Computing*, 14(4):957–970, 2021.

[19] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.