RESTgym: A Flexible Infrastructure for Empirical Assessment of Automated REST API Testing Tools

Davide Corradini*
University of Luxembourg
Luxembourg
davide.corradini@uni.lu

Michele Pasqua
University of Verona
Verona, Italy
michele.pasqua@univr.it

Mariano Ceccato
University of Verona
Verona, Italy
mariano.ceccato@univr.it

***Abstract*—As the software engineering research community continues to propose novel approaches to automated test case generation for REST APIs, researchers face the labor-intensive task of empirically validating their methodologies and comparing them with the state-of-the-art. This process requires assembling a benchmark of case studies (notoriously difficult to find in the context of REST API testing), building and running each API, gathering competitor tools, conducting experimental testing sessions to collect effectiveness and efficiency metrics, and processing the results. These extensive engineering efforts consume time that could be otherwise spent on more research-oriented tasks.**

This paper introduces RESTgym, a flexible empirical infrastructure designed to assess the performance of REST API testing tools and facilitate comparative analysis with state-of-the-art approaches. By providing a standardized environment for comparison (currently consisting of 11 benchmark APIs and 6 state-of-the-art tools packed into containers, but easily extensible to add new APIs and tools) and an orchestration engine, RESTgym significantly reduces the time and effort required for researchers to evaluate REST API testing methodologies. The paper details the architecture and components of RESTgym and demonstrates its utility through a practical example, highlighting its potential to speed up research and development in automated REST API testing. *Video: http://tiny.cc/restgym-video*

***Index Terms*—REST APIs, Automated Testing, Benchmarking Infrastructure.**

I. Introduction

RESTful APIs are a cornerstone of modern web architectures, widely adopted for their versatility. They are essential for integrating systems and building microservices architectures, as they enable different services to communicate and share data seamlessly over the web. Ensuring the reliability and correctness of these APIs is crucial, and automated test case generation has emerged as a promising method for identifying defects and vulnerabilities.

Several approaches and tools have been proposed to automate test case generation for REST APIs [1], [2], [3], [4], [5], [6], [7]. However, validating and comparing these approaches is challenging due to the lack of a standardized benchmarking infrastructure. This requires researchers to invest significant time in identifying, acquiring, building, and executing a benchmark set of REST API case studies, a set of competitor testing tools to compare with, deploying the necessary infrastructure (including the APIs, the testing tools, and metric collection tools), scripting the experimental execution, and processing the raw data, just to assess the effectiveness and efficiency of their approaches.

To alleviate this issue, we introduce RESTgym, a novel, flexible, and scalable benchmarking infrastructure for REST API testing tools, allowing researchers and practitioners to quickly assess their novel testing approach and compare it with other state-of-the-art tools. RESTgym also provides a collection of Docker images[1] of 11 benchmark API case studies, 6 state-of-the-art testing tools, metrics collection tools, and a suite of scripts to automate the orchestration of these components. Researchers can easily build a Docker image of their testing tool, plug it into RESTgym, and quickly assess its performance in terms of effectiveness and efficiency on a set of real-world and diverse APIs. RESTgym will manage the orchestration of testing sessions, applying each testing tool to every API across multiple repetitions, while also gathering experimental results regarding the effectiveness and efficiency of the tool. Because of its container-based architecture, RESTgym can be executed on any system.

II. REST API Testing

A REST API is a web API that adheres to the REST (REpresentational State Transfer) architectural style [8], allowing web clients to access and manipulate resources and invoke remote routines by leveraging stateless operations over the HTTP protocol. REST APIs provide a uniform interface to create, read, update, delete resources identified by a HTTP URI. Such operations on resources are mapped to the HTTP methods `POST`, `GET`, `PUT` and `DELETE`, respectively. Upon receiving and processing an HTTP request that exercises a specific API operation, the REST API returns an HTTP response with the outcome of the request, called status code (e.g., `2XX` for a success; `4XX` for a client-side error; or `5XX` for a server-side error), and, possibly, a payload.

REST APIs are usually documented by using the OpenAPI[2] standard. According to such standard, an API is described by a structured file (either YAML or JSON), called *OpenAPI Specification* (OAS), that indicates how to reach the API using a URI, which authentication schema is adopted, and the details

*Affiliated with the University of Verona when this work was done.

[1] Actually, these are OCI compliant images that can also be executed by other container platforms such as Podman.

[2] https://www.openapis.org/

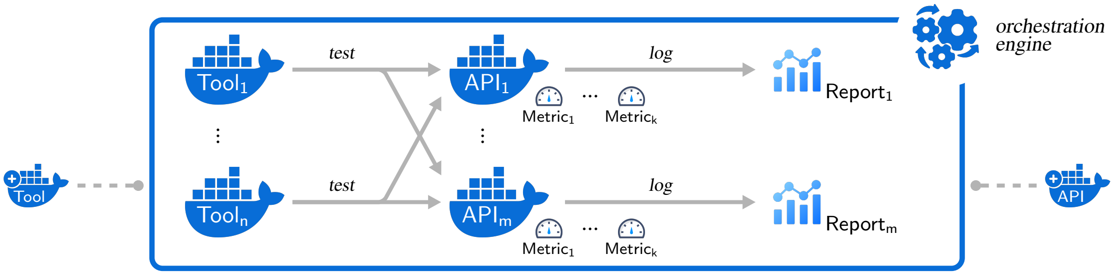

Fig. 1. RESTgym overview.

of the API available operations: the input parameters (and their schema) to be used in requests and the schema of responses.

Testing a REST API consists of crafting suitable input values for parameters of the operations exposed by the API and individuating the correct order of operations to test, to execute meaningful business logic scenarios. Currently, there are two main methods to test REST APIs: *white-box* and *black-box*. The former uses the source code of the API user test to guide test generation, while the latter approach relies solely on the information provided in the API documentation (OAS) for test case generation and interacts with the API only through the REST (HTTP) interface. Having access to the API source code is not always a viable assumption. Hence, black-box is the most adopted strategy when testing REST API, as demonstrated by a large number of black-box testing tools [9]. On the other side, there is only one tool (EvoMaster [7]) in literature performing white-box testing of REST APIs.

III. RESTgym

RESTgym is an infrastructure designed for the empirical assessment of REST API testing tools, aimed at simplifying the validation of these tools and facilitating performance comparison with competitor state-of-the-art tools. It includes a collection of Docker images for 11 benchmark REST APIs, as well as Docker images for 6 state-of-the-art tools, that can be extended with supplementary APIs and tools.

As shown in Figure 1, the RESTgym's orchestration engine automates the execution of testing sessions for each tool across all APIs, conducting multiple repetitions to account for non-determinism, while collecting experimental data about effectiveness and efficiency. Furthermore, it manages the parallel execution of testing sessions to reduce execution time while monitoring the host machine's available resources to prevent saturation. It conducts runtime health checks to verify that both the API and the testing tool containers stay alive during the testing sessions. Additionally, RESTgym performs integrity checks on completed testing sessions and re-runs any that are found to be corrupted. Finally, it compiles comprehensive reports for each experimental testing session, as well as a cumulative report summarizing all the testing sessions.

The following subsections describe in more detail how RESTgym works, and introduce the collection of REST API testing tools, benchmark API case studies, and evaluation metrics already present in RESTgym.

A. Benchmarking Testing Tools with RESTgym

When RESTgym is launched, users are guided through a step-by-step wizard in the command line interface.

Initially, RESTgym handles the building of Docker images for all tools and APIs. Images for the tools and the APIs that are natively included with RESTgym are downloaded from the Docker Hub platform [10]. Conversely, Docker images for user-provided tools and APIs are built locally on the user's machine. *We encourage RESTgym users to share their images to help us expand and enhance the infrastructure.*

Afterward, the experiment execution can begin: the user is prompted to specify the number of repetitions for each experimental testing session and the duration of each experiment. RESTgym allows for repeating the same experiment multiple times to observe various instances. This repetition aims to control the randomness of non-deterministic components in the tools and APIs. The total execution time of the experiment depends on the specified duration for each experiment, the number of repetitions, and the availability of resources (CPUs and RAM) on the host machine. RESTgym parallelizes the execution of experiments, utilizing up to 80% of the available resources to minimize the cumulative execution time. Since some APIs or tools might be unstable, each experimental testing session is monitored by the RESTgym orchestration engine. In the event of failures that cause containers to stop, the testing session is suspended and then restarted to ensure the continuity and reliability of the experiment.

When the experiment execution is complete, the user can initiate a validation step to verify the integrity of the collected experiment data. During this step, RESTgym checks that certain properties hold true for the data. For example, it ensures that metrics were consistently collected throughout the experiment, verifies that an adequate number of requests were recorded by the proxy, and confirms that coverage samples are always increasing (as coverage cannot decrease). If any of these assertions fail for some experiments, the user is prompted to delete the corrupted execution and decide whether to reschedule the execution of the compromised experiments.

Finally, once all executions have been completed and validated, the final phase involves processing the raw data to extract measures of effectiveness and efficiency. RESTgym performs this task, generating a report for each execution as well as a cumulative report summarizing all executions.

B. Testing Tools in RESTgym

Currently, RESTgym includes the following 6 state-of-the-art REST API testing tools.

RESTler [6] A stateful tool that enumerates sequences of operations according to producer-consumer relations.

RestTestGen [5] A tool that uses an Operation Dependency Graph to prioritize API operations calls based on data dependencies.

Schemathesis [3] A tool that performs property-based testing and detects faults by checking response compliance.

Morest [4] A tool that models the API under test with a Property Graph leveraged to build API invocations.

ARAT-RL [2] The first tool adopting reinforcement learning to prioritize API operations to test.

DeepREST [1] A recent approach that leverages deep reinforcement learning to guide test generation.

These tools were selected from the various options proposed in recent years due to their demonstrated effectiveness in comparative studies [11], [12], [2], [1]. If a tool implements different testing strategies, as is the case with many tools like RESTler and RestTestGen, we configured the containers to launch the default testing strategies that the tools would typically execute when downloaded from the authors' official repositories. These default strategies correspond to their most effective strategies, according to the related papers.

Despite its popularity and effectiveness, we chose not to include EvoMaster [7] among the tools available in RESTgym for two primary reasons: (i) EvoMaster operates from a white-box perspective for test generation, which we believe makes it inappropriate to compare with the other black-box tools, and (ii) the current architecture of RESTgym, which executes tools and APIs under test in separate containers, does not facilitate the operation of a white-box tool. However, we intend to support the black-box version of EvoMaster in future releases.

Users of RESTgym can contribute their own testing tools by packaging them into a Docker image. To ensure compatibility with the infrastructure, the tool must be configurable via environment variables. Specifically, the path to the OpenAPI specification of the API under test and the port on which the API responds must be adjustable through environment variables within the tool's container. Technical details on how to prepare a compatible testing tool Docker image are available in our GitHub repository [13].

C. REST APIs in RESTgym

The benchmark APIs currently available in RESTgym have been sourced from a previous study, specifically the ARAT-RL paper [2], which, in turn, include some APIs from the EvoMaster Benchmark [14]. We extend our gratitude to the authors of ARAT-RL and EvoMaster Benchmark for sharing their replication packages and artifacts, from which we selected the APIs. To this collection, we added an additional API from GitHub [15], which enhances diversity of our benchmark in terms of complexity and business logic domains. Moreover, we plan to include further APIs from EvoMaster Benchmark.

TABLE I
THE BENCHMARK APIS IN RESTGYM.

API	# Ops	LoC (JaCoCo)	LoC (IntelliJ)
REST Countries	22	543	1,121
User Management	22	632	1,284
Market	13	2,206	5,543
Project Tracking System	59	1,298	3,613
Features Service	18	457	956
NCS	6	275	500
SCS	11	295	586
Genome-Nexus	23	4,831	15,541
Person Controller	12	179	522
Blog	52	1,188	3,725
LanguageTool	2	45,487	83,708

We containerized all APIs with the necessary dependencies, setup scripts, and metric collection tools, to support RESTgym. The list of API case studies of RESTgym is reported in Table I. The table also shows the number of operations exposed by each API and the number of lines of code measured by JaCoCo [16] and the IntelliJ IDEA IDE. The size and complexity of this benchmark are consistent with existing literature on REST API testing, making it suitable for assessing novel tools and performing comparative studies. Furthermore, we believe it is representative of real-world APIs because the included case studies exhibit the following characteristics: they vary in size (in terms of the number of exposed operations and parameters), they manage resources from different domains (e.g., geography, languages, biology, etc.), and have varying levels of business logic complexity.

Users of RESTgym can contribute their own APIs in the same way they contribute tools. A Docker image of the API must be provided, and the API should respond on TCP port 9090 (specifically, behind a reverse proxy that responds on port 9090). The API image should include all necessary dependencies for the API to run properly, such as databases and libraries, as well as the metric collection tools. Further technical details are available on GitHub [13].

D. Evaluation Metrics in RESTgym

During the execution of the experimental testing sessions, RESTgym collects *effectiveness* and *efficiency* metrics, which are standard metrics commonly gathered in the majority of literature on white-box and black-box REST API testing [9].

To measure the tools' effectiveness, RESTgym collects code coverage, operation coverage, and the number of unique faults.

Code Coverage The extent to which the source code of a REST API is executed by a testing tool. RESTgym collects line, branch, and method coverage via the JaCoCo library [16] (all APIs available so far are written in Java).

Operation Coverage The extent to which API operations are successfully executed by the testing tool. An operation is defined as the combination of an HTTP method and a path, consistent with the OpenAPI standard. For instance, `GET /comment`, `POST /comment`, and `GET /comments` represent three distinct example operations. An operation is considered covered when the testing tool can generate at least one successful interaction with the operation, indicated by the API responding with a

2XX status code. A reverse proxy monitors the traffic occurring between the API and the testing tools and logs the covered operations.

Unique Faults In the REST API testing context, a tool is considered to have detected a fault when it can trigger an API response with a status code in the 5XX class (i.e., an internal server error, typically caused by an exception in the execution). In the literature, a fault is considered unique when its error message is sufficiently different from other faults observed in the API. A reverse proxy monitors the HTTP traffic and collects the error messages in the payloads of 5XX responses, and eventually computes the magnitude of unique error messages, i.e., unique faults. The bucketing algorithm was sourced from a previous study [2] and is based on the Jaccard similarity index [17]. To make it more accurate, we use a custom similarity threshold for each API that has been manually tuned on the observed error messages of each API.

To measure the efficiency of a tool, RESTgym samples the effectiveness metrics over time and in relation to the number of requests sent by the tool. This allows for the examination of trends in code coverage, operation coverage, and the number of detected faults over time, enabling insights about efficiency to be inferred from the data.

While these metrics are commonly referenced in the literature on REST API testing [9], RESTgym offers the flexibility to incorporate additional metrics derived from the raw data collected (i.e., code coverage and the history of HTTP interactions). Metrics that rely on different types of raw data may require a more extensive integration within the infrastructure.

IV. Running Example

This section introduces a running example showing how we adopted RESTgym to validate DeepREST and compare its performance with 5 competitor state-of-the-art tools for REST API testing. Additionally, we show how we added *Blog* [15] to the collection of API case studies.

A. Containerization of DeepREST

Adding a testing tool to RESTgym requires providing its Docker image. To containerize a tool, it is necessary to include the tool itself and its required dependencies within the container. For DeepREST, this involved incorporating the Stable Baselines3 Python library [18].

Additionally, the containerized tool must "expose" a control interface that is compatible with RESTgym, which uses environment variables to set the configuration of the testing tool container. To achieve this and to avoid modifying the tool itself to support RESTgym's environment variables, we included a script within the container. This script reads the environment variables and subsequently launches DeepREST with the appropriate configuration.

B. Containerization of the Blog API

Containerizing an API for RESTgym requires including not only the API executable and its dependencies, such as

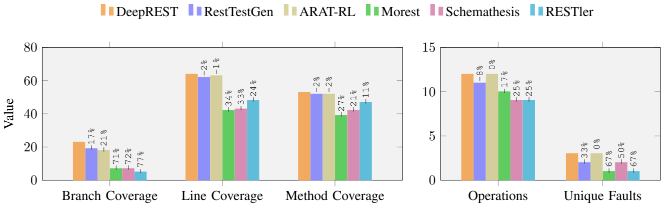

Fig. 2. Effectiveness results (aggregate) from the DeepREST paper [1].

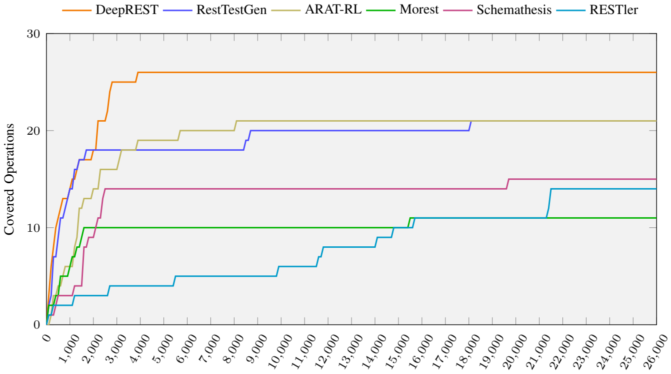

Fig. 3. Efficiency trends on the Blog API.

databases, but also all metric collection tools (e.g., JaCoCo for code coverage of Java APIs) in the image. Metrics collection tools are not deployed in a separate container because they require access to the API execution environment to collect white-box metrics, such as the source code coverage. To simplify the process of containerizing an API, RESTgym provides a comprehensive guide in the readme file, along with a Dockerfile template that has the metric collection tools already configured for contributors to draw inspiration from.

The *Blog* API is a fairly complex API that exposes 52 operations and consists of 3,725 lines of code. It was developed in Java using the Spring Boot framework, and its source code is available on GitHub [15]. After compiling the API source code and packaging it into a JAR executable, we created a Dockerfile. This Dockerfile uses a Ubuntu base image, on top of which it installs the Java Development Kit (JDK), MySQL, JaCoCo, and the MITM proxy. While the JDK and MySQL are dependencies of the Blog API, JaCoCo and the MITM proxy are employed to collect source code coverage and record HTTP interaction histories, respectively. This data is later used to extract effectiveness and efficiency metrics.

C. RESTgym Execution

After incorporating the DeepREST and *Blog* API images into RESTgym, we are ready to start the execution of the experiment. The orchestration engine downloads API and tool images from Docker Hub and builds the DeepREST and Blog API images locally. After selecting a duration of one hour for each experiment and ten repetitions for each experimental testing session, RESTgym can begin the execution. We ran RESTgym on a machine featuring a 128-core processor and 386GB of RAM, enabling many parallel testing sessions. The entire experiment, consisting of 660 hours of machine time (1 hour × 6 tools × 11 APIs × 10 repetitions), was completed in slightly less than a week. Some testing sessions failed during execution, but the health checks performed by the orchestration engine automatically detected these failures and rescheduled the executions.

After the execution is completed, results are available in a folder for each testing session. Moreover, a global CSV file is produced summarizing the final results for all executions.

D. Results

The results generated by RESTgym can be used to perform statistical analyses on the performance of testing tools, as demonstrated in the DeepREST paper [1]. For instance, Figure 2, taken from the same paper, presents a graph illustrating the cumulative results for effectiveness metrics across all executed testing sessions for DeepREST and its competitor tools. The plot reports on the left the average coverage among all 11 APIs for each tool and reports on the right the average count of successfully tested operations (2XX) and faults (5XX). See [1] for comments on the results.

Conversely, Figure 3 depicts a graph showcasing the efficiency of the tools during a single execution on the Blog API. The graph illustrates the trend in operation coverage with respect to the number of HTTP interactions with the API. Note that the graph displays the trends only up to the 26,000th request, as a plateau was reached beyond this point.

In this execution, DeepREST and RestTestGen emerged as the most efficient tools, as their performance demonstrated a steep increase during the first 2,000 interactions. However, DeepREST continues to show growth beyond that point, likely due to its novel deep reinforcement learning algorithm, and reaches the plateau quickly at the 4,000th interaction, surpassing all the other tools. Other tools exhibit a slower growth in coverage; for instance, RESTler appears to be the least efficient among them.

While RestTestGen and ARAT-RL achieve the same operation coverage by the end of the testing session, RestTestGen exhibits quicker growth initially. However, it is overtaken by ARAT-RL around the 4,000th interaction, after which both tools eventually reach the same coverage.

V. Conclusion

In this paper, we introduced RESTgym, a comprehensive and flexible benchmarking infrastructure designed to facilitate the empirical assessment of automated REST API testing tools. RESTgym addresses the engineering challenges faced by researchers in evaluating and comparing different REST API testing approaches by providing a benchmarking infrastructure that works out of the box. The infrastructure includes a diverse set of benchmark APIs and state-of-the-art testing tools, enabling thorough performance evaluations in terms of effectiveness and efficiency. This set can be easily extended by adding novel (containerized) APIs and testing tools.

Through our demonstration involving the assessment of DeepREST, we showcased RESTgym's capability to streamline empirical validations and comparative analyses among various REST API testing tools.

Future work will focus on enhancing RESTgym's features, such as including additional API case studies in the benchmark, additional automated test generation tools, and, possibly, additional metrics for more fine performance evaluations. We also plan to develop a graphical user interface to display real-time data about ongoing experiments and their results.

Data Availability

The latest version of RESTgym can be found on GitHub [13], licensed under the Apache License, version 2.0. Additionally, the version of RESTgym that was available at the time of this paper's acceptance is accessible on Zenodo [19].

References

[1] D. Corradini, Z. Montolli, M. Pasqua, and M. Ceccato, "Deeprest: Automated test case generation for rest apis exploiting deep reinforcement learning," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 2024, pp. 1383–1394.

[2] M. Kim, S. Sinha, and A. Orso, "Adaptive rest api testing with reinforcement learning," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2023, pp. 446–458.

[3] Z. Hatfield-Dodds and D. Dygalo, "Deriving semantics-aware fuzzers from web api schemas," in *Proc. of the ACM/IEEE 44th Int. Conference on Software Engineering: Companion Proceedings*, 2022, pp. 345–346.

[4] Y. Liu, Y. Li, G. Deng, Y. Liu, R. Wan, R. Wu, D. Ji, S. Xu, and M. Bao, "Morest: Model-based restful api testing with execution feedback," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 1406–1417.

[5] D. Corradini, A. Zampieri, M. Pasqua, E. Viglianisi, M. Dallago, and M. Ceccato, "Automated black-box testing of nominal and error scenarios in restful apis," *Software Testing, Verification and Reliability*, vol. 32, no. 5, p. e1808, 2022.

[6] V. Atlidakis, P. Godefroid, and M. Polishchuk, "Restler: Stateful rest api fuzzing," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 748–758.

[7] 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.

[8] R. T. Fielding, *Architectural styles and the design of network-based software architectures*. University of California, Irvine Doctoral dissertation, 2000, vol. 7.

[9] A. Golmohammadi, M. Zhang, and A. Arcuri, "Testing restful apis: A survey," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 1, nov 2023.

[10] D. Corradini, M. Pasqua, and M. Ceccato, "The Docker Hub Repositories of RESTgym," https://hub.docker.com/u/restgym, 2025.

[11] D. Corradini, A. Zampieri, M. Pasqua, and M. Ceccato, "Empirical comparison of black-box test case generation tools for restful apis," in *2021 IEEE 21st International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2021, pp. 226–236.

[12] M. Kim, Q. Xin, S. Sinha, and A. Orso, "Automated test generation for rest APIs: No time to rest yet," in *Proc. of the 31st ACM SIGSOFT Int. Symposium on Software Testing and Analysis*, 2022, pp. 289–301.

[13] D. Corradini, M. Pasqua, and M. Ceccato, "The GitHub Repository of RESTgym," https://github.com/SeUniVr/RESTgym, 2025.

[14] A. Arcuri, M. Zhang, A. Golmohammadi, A. Belhadi, O. Duman, S. Seran, J. P. Galeotti, and H. Ghianni, "The Zenodo Repository of EvoMaster Benchmark," https://doi.org/10.5281/zenodo.6106829, 2025.

[15] osopromadze. (2024) Blog REST API. [Online]. Available: https://github.com/osopromadze/Spring-Boot-Blog-REST-API

[16] E. Team, "Jacoco," https://www.eclemma.org/jacoco/, 2023.

[17] P. Jaccard, "Étude comparative de la distribution florale dans une portion des alpes et des jura," *Bull Soc Vaudoise Sci Nat*, pp. 547–579, 1901.

[18] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann, "Stable Baselines3," 2019.

[19] D. Corradini, M. Pasqua, and M. Ceccato, "The Zenodo Artifact of RESTgym," https://doi.org/10.5281/zenodo.14859630, 2025.