Systematic Literature Review: RESTful API Testing Using Machine Learning Techniques

Bill A. Otieno[1] & Eric Araka[2*]

[1]Kenyatta University, Kenya (abill2363@gmail.com)

[2]Kenyatta University, Kenya (araka.eric@ku.ac.ke)

**Corresponding author: araka.eric@ku.ac.ke*

https://doi.org/10.62049/jkncu.v5i1.449

Abstract

Representational State Transfer (REST) APIs are central to modern web and microservice systems, making their reliable testing increasingly important. Traditional automated approaches—black-box, white-box, and model-based have improved coverage but still struggle with incomplete specifications, invalid inputs, and complex workflows. Recent advances in machine learning and large language models address these issues by extracting constraints, generating realistic inputs, and guiding adaptive exploration. This review outlines how machine learning and large language models are transforming REST API testing, summarizes key techniques and empirical results, and highlights open challenges and future directions toward more intelligent, automated, and scalable RESTful API testing solutions.

Keywords*:* REST, API, RESTful, API Testing, Machine Learning, Software Testing

Introduction

Representational State Transfer (REST) has become the dominant architectural style for exposing software functionality on the web and across microservice ecosystems. RESTful Application Programming Interfaces (APIs) now underpin consumer apps, enterprise backends, and third-party integrations, making their reliability and robustness a first-order engineering concern rather than convenience. Testing these APIs, however, is uniquely challenging: requests traverse networks and intermediaries; responses depend on mutable state in databases and caches; and realistic workflows often require precise sequences of operations across many endpoints. As the number, diversity, and complexity of APIs have grown, so too has the need for automated, scalable testing approaches that can keep pace with continuous delivery practices and evolving specifications.

Traditional automated techniques for REST API testing span a spectrum from black-box to white-box. Black-box tools typically consume OpenAPI/Swagger specifications and generate requests to exercise endpoints, parameters, and status codes while monitoring failures or invariant violations. These approaches have matured into property-based test generation, fuzzing with mutation strategies, and model-based sequence exploration that learns producer–consumer dependencies among operations. White-box techniques instrument server code and data stores to guide search-based test generation toward higher coverage, and to work around chronic problems such as under-specified schemas. Collectively, these lines of work have uncovered real faults in production-grade services and improved operation/code coverage over baselines. Yet persistent pain points remain: (1) specifications often lack machine-readable constraints and inter-parameter rules; (2) generating semantically valid inputs (not just well-typed ones) is hard; (3) discovering long, stateful workflows that require precise call ordering is combinatorially explosive; and (4) test oracles frequently rely on coarse signals (e.g., HTTP 5xx) rather than rich semantic checks.

Against this backdrop, machine learning (ML)—and, more recently, large language models (LLMs)has opened new avenues. Early efforts applied natural language processing (NLP) to mine constraints and dependencies from the human-readable portions of API specifications, augmenting schemas with predicates (e.g., ranges, regex patterns) and example values that help downstream generators avoid invalid requests. Building on this idea, LLM-assisted methods now parse parameter descriptions, infer inter-parameter relationships, and synthesize realistic, context-aware inputs—often without the expensive dynamic validation loops that earlier NLP pipelines required. Other strands combine ML with planning and reinforcement learning (RL): multi-agent or value-decomposition RL coordinates choices of which operation to call next, which dependencies to satisfy, which parameter combinations to try, and which value sources to use (e.g., outputs of prior calls, LLM-generated candidates, or type-based randoms). These hybrid approaches report consistent gains in operation/code coverage and fault discovery across diverse public services (e.g., movie/music platforms) and open source backends.

The emergence of compact, fine-tuned models further broadens the design space. Instead of relying solely on large, general-purpose LLMs, recent work trains smaller models on mined corpora of REST parameters, example values, and known dependency patterns. Quantized variants reduce footprint while retaining much of the accuracy, enabling practical integration into CI/CD pipelines where latency and cost matter. Parallel advances in model-based testing continue to pay dividends: property graphs and dependency graphs help globalize a service's behavioral structure, while online feedback updates those models as tests run—bridging the gap between static specifications and live system behavior.

Despite this progress, significant open problems persist. Specifications remain incomplete and uneven in quality, requiring robust fallbacks when descriptions are vague or missing. Validating semantics beyond schemas—e.g., "does the response reflect a consistent business rule after a series of updates?"—demands oracles that go beyond status codes and JSON schema checks. Coordinating long-horizon workflows remains difficult, especially when authorization scopes, rate limits, pagination, and idempotency semantics must be respected. Finally, engineering concerns—repeatability, dataset/benchmark realism, and clear reporting on faults—are essential for technology transfer into industry.

This systematic literature review (SLR) synthesizes the research landscape at the intersection of RESTful API testing and machine learning. Our goals are to: map the evolution from traditional black-/white-box and model-based techniques to ML- and LLM-augmented approaches; analyze how ML is used (constraint extraction, value generation, dependency discovery, planning/RL, and oracles); assess empirical evidence (coverage, operation reachability, internal server errors/bug findings) and common benchmarks; and identify open challenges and promising future directions (e.g., dynamic semantic oracles, hybrid planners, small-model fine-tuning, and standard evaluation suites).

Objectives

Our objective is to synthesize how ML (including NLP and LLMs) is used to improve the automation, effectiveness, and efficiency of REST API testing. We focus on (i) what problems ML is used to solve (e.g., value generation, constraint extraction, inter-parameter dependencies, exploration/planning, oracles), (ii) how ML models and training data are configured, (iii) how these approaches are evaluated (benchmarks, metrics, baselines), and (iv) their comparative strengths, limitations, and open challenges.

Concretely, we seek to answer:

- What families of ML techniques are applied to REST API testing and at which stages of the testing pipeline do they operate?
- What model types, training data, and prompting/finetuning strategies are used?
- What improvements are reported relative to traditional methods (e.g., coverage, valid-input rates, internal error discovery)?
- What are the engineering constraints (e.g., compute cost, latency, model size), and how do studies address deployment concerns (e.g., CI/CD integration, reproducibility)?
- What persistent challenges and research opportunities remain?

Background and Related Work

RESTful API Testing Landscape

RESTful APIs have become the backbone of modern software systems, enabling interoperability across heterogeneous services and platforms (Golmohammadi, Zhang, & Arcuri, 2023). However, their reliance on network communication, databases, and third-party services introduces significant testing challenges, including input validation, dependency handling, and fault detection. Traditionally, REST API testing techniques have been categorized into black-box approaches, which rely on API specifications and observed inputs/outputs, and white-box approaches, which require access to source code and internal logic (Arcuri, Zhang, & Galeotti, 2024).

Black-box methods dominate in practice, as source code is often unavailable. Tools such as RESTler, QuickREST, and RestTestGen generate test cases from OpenAPI specifications to uncover inconsistencies, misconfigurations, and bugs (Karlsson, Čaušević, & Sundmark, 2019; Corradini, Zampieri, Pasqua, & Ceccato, 2021). However, these techniques struggle with under-specified schemas and complex inter-parameter dependencies. White-box strategies, such as search-based fuzzing with EvoMaster, offer deeper coverage but face scalability and adoption barriers in industry (Arcuri et al., 2024).

Model-Driven and Specification-Based Testing

Model-driven testing (MDT) emerged as an early attempt to automate REST API testing by leveraging abstract models of resources, states, and transitions. These models enable automatic generation of both implementation code and functional test cases (Fertig & Braun, 2015). Later, frameworks like Morest (Liu et al., 2022) introduced execution feedback to dynamically update dependency graphs, increasing fault detection and code coverage compared to earlier tools. Despite these advances, traditional specification-driven techniques are limited by their reliance on static information from OpenAPI descriptions, which often omit implicit semantic rules and constraints.

Early Use of NLP in REST API Testing

To overcome the limitations of static schemas, researchers turned to natural language processing (NLP) to extract semantic information from textual API documentation. Tools such as RESTInfer (Liu, 2022) and NLP2REST leveraged rule extraction from natural language descriptions to infer constraints and inter-parameter dependencies. Similarly, QuickREST applied property-based testing to evolve test cases as APIs changed (Karlsson et al., 2019). While effective in capturing semantic nuances, NLP-based techniques struggled with accuracy and scalability, often requiring manual validation and significant engineering effort (Martin-Lopez, Segura, & Ruiz-Cortés, 2022).

Emergence of Machine Learning for REST API Testing

Recent advances in machine learning (ML) and large language models (LLMs**)** have revitalized REST API testing research. Deep learning approaches, such as predicting valid test inputs from prior execution traces, have been shown to improve input feasibility and reduce invalid requests (Liu, Li, & Deng, 2022). Reinforcement learning has been applied to adaptively explore APIs, exemplified by APIRL for fuzzing (Zhang et al., 2022) and AutoRestTest's multi-agent reinforcement learning framework (Kim, Stennett, Sinha, & Orso, 2025).

In parallel, LLM-driven approaches such as RESTGPT (Kim, Stennett, Shah, Sinha, & Orso, 2024) and LlamaRestTest (Kim, Sinha, & Orso, 2025) demonstrated that pre-trained models can extract semantic rules, generate realistic test values, and dynamically adapt during testing. These techniques outperform traditional NLP approaches, achieving higher coverage and bug detection rates while reducing manual effort. Furthermore, frameworks like RestGPT (Song et al., 2023) highlight the potential of connecting LLMs with real-world APIs, bridging the gap between specification enhancement and automated testing.

Research Gap

Despite these advances, challenges remain. Existing ML-based solutions often focus narrowly on rule extraction or input generation, without fully integrating into end-to-end testing pipelines. Moreover, the trade-offs between model size, efficiency, and accuracy—especially in small language models versus large-

scale LLMs—require further investigation (Kim et al., 2025). This review seeks to synthesize current knowledge on RESTful API testing using ML techniques, identify gaps in scalability, adaptability, and industrial adoption, and propose future research directions.

Review Methodology

This section describes the protocol we followed to conduct a rigorous and reproducible systematic literature review (SLR) on RESTful API testing using machine learning (ML). Because the goal is an academic paper that is replicable based on our sources, we document our process with enough detail to support replication.

Sources and Corpus Construction

The primary corpus comprises 24 sampled peer-reviewed papers. These span classic black-/white-box techniques, model-driven testing, NLP-based constraint extraction, and recent LLM/SLM approaches (e.g., specification enhancement, value generation, RL-guided exploration, multi-agent planners). To reduce omission risk, we used backward snowballing (following reference lists and related-work sections) to note additional influential works for later expansion. We treat these sampled set of papers for closed world for analysis. The corpus predominantly covers 2015–2025, capturing the emergence of OpenAPI-driven black-box tools, model-based and search-based fuzzing, and the 2022–2025 wave of NLP/LLM methods. We considered English-language journal and conference papers, workshop/NIER papers that introduce novel ideas with preliminary evaluations, and credible arXiv preprints that are widely cited in the area.

Eligibility Criteria

To ensure rigor and consistency in the selection of studies, a two-stage screening process was applied. In the first stage, papers were filtered based on their titles and abstracts to remove clearly irrelevant items. In the second stage, the full texts of the remaining papers were examined against predefined inclusion and exclusion criteria. From an initial pool of approximately ninety-eight papers, thirty-four were shortlisted, and finally twenty-four were retained for detailed analysis.

Studies were included only if they met all of the following conditions. First, the research had to focus on the testing of Representational State Transfer (REST) or RESTful Application Programming Interfaces (APIs), including aspects such as test generation, execution, fault detection, oracle design, or coverage measurement. Second, the study needed to incorporate machine learning, natural language processing, or large language models at any stage of the testing process—whether for constraint or value extraction, dependency inference, exploration and planning, dynamic adaptation, or oracle learning. Third, the study had to present empirical evidence, with evaluations conducted on open-source systems, industrial or online APIs, or established benchmarks. Position or New Ideas and Emerging Results (NIER) papers were considered only if they proposed a concrete algorithm and included at least a small-scale experimental validation.

Papers were excluded if they did not meet these conditions. Specifically, studies were removed if they were unrelated to REST (for example, those focused solely on SOAP or GraphQL) unless their methods were explicitly transferable and validated on REST APIs. Similarly, tool descriptions lacking empirical evaluation or any machine learning or artificial intelligence component were excluded, as were surveys and vision papers that did not propose new methods—though these were retained as contextual references. Non-English works, duplicates, or studies without accessible full texts were also excluded.

The final set of papers represented a diverse range of approaches. These included black-box testing methods such as QuickREST, RestTestGen, and swarm-intelligence-based testing; white-box strategies like EvoMaster enhanced with advanced heuristics; model-based techniques with execution feedback such as Morest; natural language processing approaches for constraint extraction, exemplified by RESTInfer; reinforcement learning-based fuzzing; and the most recent methods driven by large and small language models, including RESTGPT, LlamaRestTest, and multi-agent or semantic-graph frameworks. Survey and methodological papers were referenced mainly to provide context and background but were not treated as primary evidence for assessing testing effectiveness.

Data Extraction and Coding

A structured data extraction sheet was used to collect consistent information from each study. It included bibliographic details such as authors, publication venue, and year, along with methodological elements aligned with the review's research questions. For each paper, we recorded the testing focus (for example, specification enhancement, input generation, dependency discovery, or oracle design), the type of technique used (such as large or small language models, reinforcement learning, or hybrid methods), and how the learning component was integrated with a test generator like RESTler, RestTestGen, EvoMaster, or Morest.

We also captured details of the evaluation setup, including the systems under test (open-source or online), the number of APIs and endpoints, and whether authentication or rate limits were involved. Key metrics such as code and operation coverage, valid-input rate, fault detection, constraint extraction accuracy, and efficiency were extracted, along with baselines, reproducibility information, and any threats to validity noted by the authors.

Data extraction was performed in two stages: an initial calibration on a small subset of papers to align coding decisions, followed by a full extraction pass. Any disagreements were resolved through discussion and review of the relevant sections.

Quality Appraisal

To ensure the quality of evidence and avoid simple vote counting, each primary study was appraised across five dimensions. We examined the rigor of evaluation, including the clarity of research questions, the realism of datasets and baselines, and the use of appropriate metrics or statistical analysis where needed. Reproducibility was assessed based on the availability of code, data, configurations, and versioned specifications. Construct validity considered whether reported metrics matched the study's claims, while external validity evaluated the diversity of APIs tested and the inclusion of both open-source and online services. Finally, practicality was judged by factors such as computational cost, model size, integration feasibility, and adherence to ethical testing practices. These criteria were applied qualitatively during synthesis, with stronger studies highlighted and potential threats to validity noted for discussion.

Synthesis Strategy

Because the studies included diverse methods and metrics, we adopted a narrative synthesis supported by structured comparisons and limited quantitative summaries. Coverage and operation reach are reported in terms of central trends and relative improvements over baselines rather than raw figures. For constraint and value generation, we highlight reported precision, recall, and valid-input rates, noting where improvements are most evident. Fault-finding results are summarized by the number of unique internal errors or confirmed

bugs, while efficiency is discussed in terms of operations per run and cost–accuracy trade-offs from smaller or quantized models. We also consider ablation results to identify the added value of individual components, such as specification enhancement or feedback mechanisms. When definitions vary across studies, we describe them consistently and flag non-comparable cases to avoid misleading conclusions. Tables and figures will be added later to present detailed per-study metrics.

Validity Considerations for the Review

Several factors may affect the validity of this review. Selection bias is possible since the corpus includes mainly recent and relevant studies, potentially under-representing negative results or industrial experiences; to mitigate this, backward snowballing was used to identify missing key works. Publication bias may favor studies reporting positive machine learning results, so greater weight was given to those with transparent ablations and strong baselines. Construct bias arises from inconsistent metrics across REST API testing studies, which we address by analyzing trends in effect direction rather than pooled values. Recency bias is also a concern, as many large language model papers from 2023–2025 are preliminary or based on limited data; these cases are explicitly noted. Finally, we account for ethical and legal constraints in online testing by distinguishing between studies conducted in sandboxed settings and those using live services subject to rate limits and terms of service.

Traditional and Heuristic Approaches

Before the introduction of machine learning and large language models into RESTful API testing, a variety of traditional and heuristic approaches were developed. These approaches laid the foundation for subsequent research by offering automated means of generating test cases, exploring service behaviors, and maximizing fault detection. They can be grouped into three broad categories: black-box testing tools, swarm intelligence and evolutionary algorithms, and model-based testing.

Black-Box Testing Tools

Black-box approaches have historically dominated RESTful API testing because access to source code is often unavailable in industry practice (Golmohammadi, Zhang, & Arcuri, 2023). Instead, these methods rely on service specifications such as the OpenAPI Specification (OAS) to generate requests and observe outputs. Among the most prominent black-box tools are RESTler, RestTestGen, QuickREST, and bBOXRT.

RESTler, developed at Microsoft, systematically generates request sequences from OAS definitions. It employs a feedback-driven approach where failures in earlier test runs inform subsequent request generation. Studies show that RESTler excels at robustness and discovering complex bugs in industrial services by exploring deep request chains (Atlidakis, Godefroid, & Polishchuk, 2019; Martin-Lopez, Segura, & Ruiz-Cortés, 2022). However, its bottom-up construction of request sequences may lead to scalability challenges in very large APIs (Liu et al., 2022).

RestTestGen was proposed to provide a modular and extensible framework for building black-box testing strategies. Its architecture includes reusable components such as parsers, mutation operators, value generators, and oracles (Corradini, Zampieri, Pasqua, & Ceccato, 2021). By offering a flexible foundation, RestTestGen reduces the engineering burden of re-implementing common testing primitives, thereby accelerating research on new black-box testing methods. Empirical results suggest that RestTestGen

achieves higher operation and endpoint coverage than RESTler when applied to certain open-source APIs (Golmohammadi et al., 2023).

QuickREST introduced property-based testing to RESTful APIs. Instead of generating static test cases, it creates properties from OpenAPI descriptions and evolves test cases automatically when API specifications change (Karlsson, Čaušević, & Sundmark, 2019). This makes QuickREST particularly adaptable in agile development environments where APIs evolve frequently.

Other tools such as bBOXRT combine fuzzing and replay mechanisms, using API traffic to guide the generation of new test cases. Although bBOXRT and similar tools can uncover robustness issues, they often suffer from under-specified schemas that limit coverage (Arcuri, Zhang, & Galeotti, 2024).

Comparative studies confirm that while RESTler is strong in robustness and fault detection, RestTestGen provides higher flexibility and coverage, and QuickREST demonstrates adaptability through property evolution (Ahmed, Abuhaliqa, Catal, & Mishra, 2022; Golmohammadi et al., 2023).

Swarm Intelligence and Evolutionary Algorithms

Another major line of research employs search-based and bio-inspired algorithms to optimize test suite generation. These approaches treat REST API testing as a combinatorial search problem, where the goal is to maximize structural coverage while minimizing redundant or infeasible tests.

Ahmed and Hamdy (2022) proposed the use of the Artificial Bee Colony (ABC) algorithm to automate black-box testing based on OpenAPI specifications. Their approach generates both nominal tests, which conform to the documented schema, and mutation tests, which deliberately deviate from specifications. This dual strategy achieves higher coverage across paths, operations, parameters, and status codes, exposing not only expected behaviors but also exceptional flows.

The most mature evolutionary approach is EvoMaster, which combines white-box and black-box testing through evolutionary search. EvoMaster leverages instrumentation to collect execution data and applies search heuristics to maximize line, branch, and method coverage (Arcuri et al., 2024). Recent extensions introduced advanced heuristics for handling under-specified schemas and SQL database constraints, improving robustness in real-world services. Independent studies confirm that EvoMaster consistently ranks among the top performers in tool comparisons, detecting thousands of confirmed bugs across industrial APIs (Golmohammadi et al., 2023; Arcuri et al., 2024).

Swarm intelligence and evolutionary algorithms demonstrate that heuristic optimization can handle the complexity of RESTful API search spaces. However, they often require significant computational resources, and their performance may degrade in the presence of poorly specified or incomplete API documentation.

Model-Based Testing

A third trajectory involves model-based testing (MBT), which abstracts REST APIs as models of resources, states, and transitions. These models provide a systematic basis for generating functional and security tests.

Early work by Fertig and Braun (2015) demonstrated how model-driven testing could automatically generate both API code and corresponding test cases from high-level models. This approach reduced manual effort and ensured alignment between implementation and testing.

More recently, Morest advanced MBT by introducing RESTful-service Property Graphs (RPGs) that are dynamically updated with execution feedback (Liu et al., 2022). Morest combines the strengths of top-down and bottom-up approaches: it models producer–consumer dependencies among operations, then adapts the graph during runtime based on actual responses. Empirical evaluations show that Morest outperformed both RESTler and RestTestGen, achieving up to 103% more-line coverage and detecting up to 215% more bugs across real-world services, including Bitbucket.

Other model-driven approaches, such as MDD4REST (Deljouyi & Ramsin, 2022), emphasize model-driven development (MDD) principles for RESTful APIs and demonstrate that test cases can be automatically derived as a by-product of model transformations. These works illustrate the synergy between software modeling and testing, although their industrial adoption is limited by the need for high-quality models and domain-specific expertise.

Comparative Insights and Limitations

Collectively, these traditional and heuristic methods established the methodological backbone of RESTful API testing. Black-box tools like RESTler and QuickREST offered practical automation for industry, while RestTestGen provided an extensible foundation for academic research. Swarm intelligence methods, including ABC and EvoMaster, showed that optimization techniques can push coverage and fault discovery beyond naive fuzzing. Model-based techniques such as Morest demonstrated that maintaining a semantic graph of dependencies significantly improves exploration of complex workflows.

Nevertheless, limitations persist. Black-box tools often struggle with under-specified schemas, leading to invalid inputs and shallow coverage. Evolutionary approaches can be resource-intensive and sensitive to search heuristics. Model-based methods require accurate models and risk degradation if API specifications are poorly documented. Moreover, most traditional approaches rely on simple oracles such as HTTP status codes, which provide limited insight into semantic correctness. These shortcomings created the conditions for the emergence of machine learning techniques, which aim to enhance specification analysis, input generation, dependency discovery, and oracle inference in REST API testing.

NLP and Early ML-Based Approaches

The limitations of traditional specification-driven tools—particularly their inability to handle incomplete or under-specified schemas—created a research space for techniques that leverage natural language processing (NLP) and early machine learning (ML) to enrich REST API testing. Since REST APIs are often accompanied by textual descriptions embedded in OpenAPI specifications or developer documentation, NLP methods were applied to extract semantic rules, parameter constraints, and inter-parameter dependencies that could not be captured through type signatures alone. These early efforts mark the transition from purely heuristic approaches toward data-driven testing, laying the groundwork for later large language model (LLM)-based enhancements.

RESTInfer: Constraint Extraction from Textual Descriptions

One of the most influential early NLP-driven approaches was RESTInfer, introduced by Liu (2022). RESTInfer proposed a two-phase pipeline that automatically infers parameter constraints from natural language descriptions found in OpenAPI specifications. By parsing textual cues such as "must be earlier than," "should equal," or enumerations embedded in descriptions, RESTInfer generated logical propositions that could then guide input generation. For example, it could infer that a parameter start_date must precede end_date or that a storage parameter should only take the values 128 or 256.

Evaluation results showed that RESTInfer significantly improved feasibility of test inputs, leading to higher code coverage and bug discovery compared to random generation (Liu, 2022). While preliminary, this work demonstrated the viability of using NLP for bridging the gap between human-readable descriptions and machine-executable test cases.

NLP2REST: Extracting Dependencies and Constraints

Building on this paradigm, NLP2REST explored a more generalizable NLP framework for extracting inter-parameter dependencies from specifications (Martin-Lopez, Segura, & Ruiz-Cortés, 2022). The tool employed syntactic and semantic analysis to map natural language into structured OpenAPI keywords, such as minimum, maximum, or pattern, which are required for automated test generation. To validate extracted constraints, NLP2REST combined rule extraction with dynamic checks against running APIs, ensuring that rules not only matched descriptions but also held during execution.

Although effective, the reliance on dynamic validation introduced computational overhead and limited scalability. In addition, NLP2REST often struggled with ambiguous or poorly written documentation, leading to noisy or incomplete constraint extraction (Martin-Lopez et al., 2022). Nonetheless, its hybrid use of NLP with runtime validation marked a significant methodological step toward semantic-aware REST API testing.

ARTE: Leveraging External Knowledge Bases

In parallel, researchers developed ARTE (Automatic REST Testing with External Knowledge), which enriched parameter value generation by combining NLP with semantic knowledge bases such as DBPedia. ARTE used parameter names and descriptions to query knowledge graphs, producing candidate values that were more realistic than purely random or combinatorial inputs. For instance, a parameter labeled country could yield actual country names retrieved from DBPedia, thereby increasing the likelihood of generating valid and semantically meaningful requests.

While innovative, ARTE suffered from accuracy and relevance limitations, as knowledge base lookups sometimes returned irrelevant or inconsistent values. The approach also depended heavily on parameter naming conventions, reducing robustness when APIs used domain-specific or non-standard terms. Despite these limitations, ARTE demonstrated the potential of combining external semantic resources with NLP-driven constraint inference, pushing REST API testing closer to semantic correctness.

Contributions and Limitations of Early NLP/ML Approaches

Together, RESTInfer, NLP2REST, and ARTE demonstrated how machine learning and natural language processing can enhance API specifications by uncovering hidden semantic information. These approaches

advanced the field in three main areas: RESTInfer focused on discovering and formalizing logical parameter constraints; NLP2REST improved the handling of inter-parameter dependencies to generate more valid multi-parameter requests; and ARTE enriched input generation by drawing on external knowledge sources. Despite these advances, their limitations highlighted the need for more powerful models. Constraint extraction accuracy often fell short of practical thresholds without costly validation, natural language ambiguity remained difficult to resolve, and scalability was limited by dependence on external data and complex validation workflows.

Paving the Way for LLM-Based Enhancements

Despite their shortcomings, these early NLP and ML-based tools established the foundation upon which LLM-powered systems such as RESTGPT (Kim, Stennett, Shah, Sinha, & Orso, 2024) and LlamaRestTest (Kim, Sinha, & Orso, 2025) were later built. By demonstrating that valuable semantic information could be mined from natural language descriptions, RESTInfer, NLP2REST, and ARTE paved the way for more context-aware, fine-tuned models capable of extracting richer rules, generating valid parameter values without extensive validation, and dynamically adapting to API responses. These early approaches can thus be seen as critical stepping stones between traditional heuristic testing and the data-driven LLM-based testing revolution.

LLM-Based Advances

Building on the lessons of NLP-driven tools like RESTInfer and NLP2REST, researchers have begun to exploit the power of large language models (LLMs) for RESTful API testing. LLMs offer two critical advantages: (1) their ability to capture semantic relationships from natural language descriptions without extensive manual feature engineering, and (2) their adaptability in generating realistic, contextually appropriate test inputs. Recent advances show that LLMs can improve constraint extraction, parameter value generation, dependency discovery, and planning for multi-step API interactions. This section synthesizes four major lines of work: RESTGPT, the RestGPT framework, AutoRestTest, and LlamaRestTest.

RESTGPT: Augmenting Specifications with LLMs

RESTGPT was introduced by Kim, Stennett, Shah, Sinha, and Orso (2024) as a lightweight but powerful approach to augmenting REST API specifications with constraints and example values extracted by LLMs. Unlike earlier NLP tools such as NLP2REST, which required expensive validation loops to confirm constraints, RESTGPT relied on the contextual awareness of LLMs to infer rules directly from natural-language descriptions embedded in OpenAPI specifications.

RESTGPT's workflow consisted of two key stages: (a) extracting machine-interpretable rules (e.g., ranges, enumerations, ordering constraints), and (b) generating realistic parameter values aligned with these rules. The augmented specifications could then be fed into conventional black-box testing tools, immediately improving their coverage and bug-finding capabilities.

Empirical results demonstrated a substantial boost in accuracy: RESTGPT achieved a 97% precision rate in constraint extraction, compared to 50% without validation and 79% with validation for NLP2REST (Martin-Lopez et al., 2022). Moreover, RESTGPT generated syntactically and semantically valid inputs for 73% of parameters, significantly outperforming ARTE, which only achieved 17% validity. These results

confirmed that LLMs could achieve state-of-the-art performance without costly validation pipelines, making them practical for continuous integration settings.

RestGPT Framework: Connecting LLMs to Real-World APIs

While RESTGPT aimed to enhance API specifications, the RestGPT framework extended the application of large language models (LLMs) to direct interaction with real-world APIs. Developed by Song et al. (2023), RestGPT introduced a *coarse-to-fine planning mechanism* composed of three modules: a Planner, an API Selector, and an Executor. The Planner decomposes complex user instructions into smaller, manageable sub-tasks; the API Selector then maps these sub-tasks to specific API operations based on OpenAPI documentation; and the Executor invokes the selected APIs, generates parameters, and parses responses through schema-aware code.

This iterative, online planning process enables RestGPT to dynamically adapt to API responses, allowing it to handle real-world variability with greater robustness. In evaluations using the RestBench benchmark—featuring domains such as the TMDB movie database and Spotify music services—RestGPT successfully executed multi-step workflows involving dependent API calls. When compared to tool-augmented LLM systems like Toolformer and ReAct, RestGPT exhibited stronger robustness, task decomposition, and response parsing capabilities, establishing a foundation for LLM-driven testing in live service environments (Song et al., 2023).

AutoRestTest: Multi-Agent Reinforcement Learning with LLM Inputs

Moving beyond single-agent designs, AutoRestTest introduced by Kim, Stennett, Sinha, and Orso (2025) proposed a multi-agent framework that integrates reinforcement learning (RL), semantic dependency graphs, and large language model (LLM)–generated inputs. The system coordinated several specialized agents—one for selecting the next API call, another for managing dependencies, and a third for generating input values.

The central innovation of AutoRestTest lies in its fusion of LLMs with semantic graphs, allowing agents to reason more effectively about inter-parameter relationships and call sequencing. Reinforcement learning optimized collaboration among agents, enabling the framework to adapt dynamically to runtime feedback during testing.

Empirical evaluations on Spotify and TMDB APIs demonstrated that AutoRestTest outperformed leading black-box tools such as RESTler, Morest, and RestTestGen in both operation coverage and fault detection. It was also the only approach to uncover previously undetected internal errors in Spotify APIs, underscoring its robustness in real-world environments. Moreover, the system achieved a balance between exploration—through novel input generation—and exploitation—through semantically guided requests—via agent cooperation.

By combining multi-agent reinforcement learning with LLM-driven reasoning, AutoRestTest showed that REST API testing can be both adaptive and collaborative, achieving greater coverage and fault discovery than traditional single-tool methods (Kim et al., 2025).

LlamaRestTest: Small Language Models for Efficient API Testing

Although RESTGPT and AutoRestTest showcased the potential of large-scale large language models (LLMs), issues related to computational cost, inference latency, and integration within continuous integration and continuous deployment (CI/CD) pipelines persisted. To overcome these challenges, LlamaRestTest was developed by Kim, Sinha, and Orso (2025) to explore whether smaller, fine-tuned language models could match or surpass the performance of larger ones when trained on REST-specific data.

LlamaRestTest fine-tuned and quantized the Llama3-8B model using a comprehensive dataset of REST API parameter examples and inter-parameter dependencies. The framework introduced two specialized sub-models: LlamaREST-IPD, which focuses on identifying inter-parameter dependencies, and LlamaREST-EX, which generates realistic parameter values. These models were embedded within a reinforcement learning framework to enable adaptive behavior during testing.

Empirical results demonstrated that LlamaRestTest achieved a 72.4% success rate in valid input generation—surpassing both the base Llama3-8B model (22.9%) and RESTGPT (68.8%)—and correctly identified 12 of 17 inter-parameter dependencies, compared to 9 for RESTGPT. It also outperformed established tools such as EvoMaster, Morest, and ARAT-RL across line, branch, and method coverage metrics. Notably, the model maintained strong performance under 4-bit and 8-bit quantization, supporting efficient deployment with minimal computational overhead.

Overall, LlamaRestTest illustrates that smaller, fine-tuned models can outperform general-purpose LLMs in specialized REST API testing tasks, offering an effective balance between accuracy, adaptability, and efficiency (Kim et al., 2025).

Synthesis of LLM-Based Advances

Together, these four approaches represent a major paradigm shift in the testing of Representational State Transfer (REST) APIs. RESTGPT demonstrated that large language models (LLMs) can effectively enhance API specifications by generating accurate rules and realistic input values. The RestGPT framework (Song et al., 2023) extended this capability, showing that LLMs can plan and execute real-world API workflows through coordinated task decomposition and adaptive execution. AutoRestTest further advanced the field by combining multi-agent reinforcement learning with LLM-based input generation, achieving significant gains in both coverage and fault discovery. Finally, LlamaRestTest (Kim, Sinha, & Orso, 2025) proved that smaller, fine-tuned models can rival or even surpass large-scale LLMs while offering greater efficiency and suitability for deployment.

Across these studies, LLMs consistently improved constraint extraction, dependency handling, valid input generation, and multi-step workflow execution. However, ongoing challenges remain in managing the trade-offs between computational cost and accuracy, ensuring reproducibility, and integrating these advanced testing techniques smoothly into industrial continuous integration and continuous deployment (CI/CD) pipelines.

Empirical Insights and Benchmarks

An essential dimension of RESTful API testing research lies in its empirical validation. Because automated testing approaches must ultimately prove their value through real-world coverage, fault detection, and scalability, much effort has gone into designing benchmarks, conducting large-scale studies, and comparing tools across diverse APIs. This section synthesizes key empirical insights and highlights the role of benchmarks in shaping the field.

Large-Scale Online Testing Studies

Among the most ambitious empirical efforts is RESTest, an online testing platform designed to continuously evaluate RESTful APIs at scale (Segura et al., 2018; Martin-Lopez et al., 2022). RESTest was deployed on 19 industrial APIs, including services from YouTube, Amadeus, and Slack, and generated more than one million test cases over an extended campaign (Segura et al., 2018). This large-scale evaluation led to the discovery of 254 reproducible faults, many of which were confirmed by service providers and traced to specification mismatches, undocumented constraints, or edge-case handling errors. Importantly, the faults were not limited to internal server errors but also included semantic inconsistencies, such as parameter combinations that produced invalid yet non-crashing responses.

The RESTest study demonstrated the scalability of automated REST API testing and provided one of the first systematic datasets of real-world faults. However, it also underscored challenges such as rate limiting, authentication barriers, and the ethical concerns of testing live APIs without disrupting production services (Martin-Lopez et al., 2022).

Tool Comparisons and Benchmarks

Several empirical studies have compared state-of-the-art tools across open-source services and controlled benchmarks. For example, Golmohammadi, Zhang, and Arcuri (2023) conducted a systematic evaluation of black-box test generation tools, including RESTler, RestTestGen, and QuickREST, reporting that while RESTler consistently excelled in robustness and bug detection, RestTestGen achieved higher endpoint and request coverage, and QuickREST demonstrated adaptability through property evolution. Similarly, Liu et al. (2022) showed that Morest outperformed both RESTler and RestTestGen, with up to 103% more-line coverage and 215% more bug discoveries, highlighting the power of model-based approaches with execution feedback.

Benchmarking also plays a central role in evaluating ML-based approaches. The RestBench dataset, introduced alongside the RestGPT framework (Song et al., 2023), provided a curated collection of real-world APIs from domains such as movies and music (TMDB and Spotify) with annotated tasks requiring multi-step interactions. This benchmark enabled controlled comparisons between LLM-based planners and earlier tool-augmented LLMs like Toolformer and ReAct, demonstrating RestGPT's superior robustness and accuracy. Similarly, LlamaRestTest (Kim, Sinha, & Orso, 2025) was evaluated on a diverse set of APIs, confirming that small, fine-tuned models outperformed RESTGPT and other baselines in constraint extraction, valid input generation, and fault detection.

Surveys and Meta-Analyses

Complementing individual tool studies, systematic surveys provide meta-level insights. Ehsan, Abuhaliqa, Catal, and Mishra (2022) conducted one of the earliest systematic literature reviews on RESTful API testing methodologies, categorizing approaches into black-box, white-box, model-driven, and search-based fuzzing. Their findings confirm a significant growth of research interest after 2017, corresponding with the increasing ubiquity of REST APIs in cloud services and microservice architectures.

The survey also highlighted enduring challenges:

- Test oracles remain underdeveloped, with most tools relying on HTTP status codes rather than semantic validation of responses.
- Specification incompleteness limits the reliability of specification-based tools, especially when OpenAPI documents omit inter-parameter dependencies or constraints.
- Industrial adoption is hampered by scalability issues (e.g., computational cost of evolutionary fuzzing, inference costs of large LLMs) and by the difficulty of integrating research prototypes into CI/CD pipelines.

A second empirical survey by Golmohammadi et al. (2023) reinforced these findings, noting that many tools perform well on academic benchmarks but fail to handle authentication, authorization, and rate-limiting constraints in real-world APIs.

Key Insights from Empirical Evaluations

Overall, the empirical literature reveals four key insights. First, scalability and coverage improvements are evident but inconsistent—large-scale systems such as RESTest can generate millions of test cases, yet performance still depends on specification quality and service rate limits. Second, no single tool excels across all metrics: RESTler offers strong robustness, RestTestGen achieves high coverage, Morest is most effective in bug discovery, and QuickREST adapts well to diverse scenarios. Large language model–based tools lead in semantic understanding and valid input generation but remain computationally expensive. Third, benchmarking remains fragmented; although efforts like RestBench and RESTest provide structured evaluation environments, the absence of shared datasets limits reproducibility and fair comparison. Finally, industrial maturity is still developing, with few tools ready for production use, though frameworks such as RestTestGen and LlamaRestTest, particularly with quantified models, show growing potential for integration into continuous integration and delivery pipelines.

Challenges and Open Issues

Despite significant progress in automating RESTful API testing, a number of persistent challenges remain unresolved. These challenges span from fundamental specification issues to the practical constraints of deploying resource-intensive LLM-based methods in industrial contexts. The following subsections discuss the most pressing concerns identified across empirical studies, surveys, and state-of-the-art tools.

Specification Gaps and Incompleteness

A recurring obstacle across all approaches is the incompleteness and inconsistency of API specifications. Many OpenAPI documents omit important constraints, such as allowable value ranges, inter-parameter dependencies, or conditional logic. As a result, automated tools often generate invalid or semantically

meaningless requests. For example, RESTler and RestTestGen rely on producer–consumer dependency graphs extracted from specifications, but their accuracy is undermined when specifications are sparse or poorly written (Liu et al., 2022; Corradini, Zampieri, Pasqua, & Ceccato, 2021). Similarly, NLP-based approaches like RESTInfer and NLP2REST showed that extracting constraints from free-text descriptions is possible but unreliable in the presence of vague or inconsistent documentation (Martin-Lopez, Segura, & Ruiz-Cortés, 2022). In practice, the lack of standardized, high-quality specifications continues to hinder the applicability of automated testing frameworks.

Fault Oracles and Semantic Validation

Another central limitation lies in the test oracles used to assess correctness. Most tools rely on HTTP status codes or schema validation (e.g., ensuring a JSON matches its expected format). While this approach can detect crashes (5xx errors), it fails to capture deeper semantic faults, such as violations of business rules or logical inconsistencies across multiple calls. Surveys confirm that oracle weakness is a primary barrier to industrial adoption, since real-world services demand guarantees about functional correctness rather than just stability (Ehsan, Abuhaliqa, Catal, & Mishra, 2022). Research into metamorphic testing (Segura et al., 2018) and constraint-based validation (Huayao et al., 2021) shows promise, but semantic oracles remain underdeveloped compared to input generation and coverage methods.

Scalability of Industrial-Scale Testing

Industrial-scale online testing incurs significant computational costs. RESTest, for example, generated over one million test cases across industrial APIs such as YouTube and Amadeus, ultimately detecting 254 reproducible faults (Segura et al., 2018). However, the scale of execution required to achieve these results highlights scalability concerns: executing large volumes of requests against live services introduces rate-limiting issues, increases infrastructure costs, and risks service disruption. Search-based fuzzing (e.g., EvoMaster) and multi-agent RL frameworks (e.g., AutoRestTest) also require heavy computation to converge on effective strategies (Kim, Stennett, Sinha, & Orso, 2025). These challenges raise questions about the practical feasibility of deploying such methods in continuous integration and delivery (CI/CD) pipelines.

Integration of Feedback in LLM-Based Approaches

While LLMs such as RESTGPT and LlamaRestTest demonstrate superior performance in extracting constraints and generating realistic values, they still struggle to integrate live feedback dynamically. Current LLM-enhanced methods often operate on static OpenAPI descriptions, with limited adaptation based on runtime observations. By contrast, model-based tools like Morest excel at incorporating execution feedback into dependency graphs, enabling adaptive test sequence generation (Liu et al., 2022). LLM frameworks such as RestGPT and AutoRestTest have begun to address this gap through coarse-to-fine planning and reinforcement learning agents, but their ability to seamlessly adapt LLM reasoning to evolving system states remains immature. This integration challenge is particularly acute in APIs with complex workflows, where runtime dependencies cannot always be inferred from static text.

Efficiency and Deployment Constraints

Finally, efficiency and resource demands present practical barriers. Large LLMs such as GPT-4-class models deliver impressive semantic accuracy but incur high inference costs and latency, making them

unsuitable for large-scale, routine testing. This issue has motivated research into smaller, fine-tuned models such as LlamaRestTest, which demonstrated that 8B-parameter models with quantization could outperform larger models while remaining cost-effective (Kim, Sinha, & Orso, 2025). Nonetheless, striking the right balance between accuracy, efficiency, and deployment feasibility remains an unresolved problem. In resource-constrained environments, efficiency may take precedence, but this risks sacrificing the semantic richness needed to uncover subtle bugs.

Summary of Open Issues

Looking ahead, RESTful API testing is moving toward hybrid frameworks that combine model-driven, heuristic, and large language model (LLM) techniques for greater semantic and structural coverage. Future work should focus on dynamic, machine learning–based oracles for richer validation, lightweight fine-tuned models for efficient CI/CD integration, and standardized benchmarks to improve comparability and reproducibility. Emphasis on explainability will also be key to building trust in ML-driven testing. Overall, machine learning, particularly LLMs and reinforcement learning, is transforming REST API testing, but progress depends on overcoming persistent challenges in specification quality, semantic validation, scalability, and efficiency to achieve reliable, industry-ready solutions.

Future Directions

The evolution of RESTful API testing demonstrates a clear trajectory: from traditional black-box tools and heuristic fuzzing, through NLP-driven constraint extraction, to the current wave of LLM-enhanced and reinforcement learning–based approaches. Yet, as discussed in Section 8, several challenges persist. To bridge the gap between academic innovation and industrial adoption, future research must pursue integrated, efficient, and explainable solutions. Below, we outline the most promising directions.

Hybrid Approaches for Adaptive Testing

A consensus is emerging that no single technique is sufficient to cover the multifaceted challenges of REST API testing. Hybrid approaches that combine model-driven methods, search-based fuzzing, and LLM-enhanced semantic analysis are a natural next step (Liu et al., 2022; Arcuri, Zhang, & Galeotti, 2024). For instance, Morest's execution-feedback graphs could be integrated with RESTGPT's constraint extraction to provide both structural coverage and semantic validity. Similarly, evolutionary fuzzing engines like EvoMaster could incorporate LLM-generated inputs to diversify test suites while still being guided by structural coverage heuristics. By merging the strengths of complementary paradigms, hybrid frameworks may deliver both depth (workflow exploration) and breadth (parameter value diversity) in automated testing.

Dynamic Oracles with ML and Anomaly Detection

The oracle problem remains one of the weakest links in REST API testing. Future research should prioritize the development of dynamic, ML-based oracles capable of validating semantic correctness beyond status codes. Anomaly detection techniques can be applied to API response payloads to flag suspicious behaviors, such as inconsistent field values or violations of inferred invariants (Ehsan, Abuhaliqa, Catal, & Mishra, 2022). Moreover, reinforcement learning could be leveraged to train oracles that adapt over time, refining their decision boundaries based on feedback from execution traces. Such oracles would be particularly

valuable for mission-critical domains, such as financial and healthcare APIs, where functional correctness is as important as robustness.

Fine-Tuned Lightweight Models for CI/CD Integration

The success of LlamaRestTest highlights the potential of small, fine-tuned language models (SLMs) to achieve strong performance while being computationally efficient (Kim, Sinha, & Orso, 2025). Future research should explore domain-specific fine-tuning across diverse API ecosystems, quantization for efficient inference, and automated pipelines for continuously updating models as APIs evolve. Such lightweight models are especially suited for integration into CI/CD pipelines, where efficiency, repeatability, and scalability are critical. By reducing reliance on large, general-purpose LLMs, researchers can create solutions that are both practically deployable and cost-effective.

Benchmarks, Standards, and Reproducibility

Progress in REST API testing has been hampered by the fragmented evaluation landscape. While platforms like RESTest (Segura et al., 2018) and RestBench (Song et al., 2023) have provided valuable testbeds, the community still lacks standardized, widely adopted benchmarks that cover both open-source and industrial APIs. Future work should focus on curating diverse, realistic datasets that reflect modern API challenges such as authentication, rate-limiting, and workflow dependencies. Furthermore, reproducibility standards—including open-source toolchains, replication packages, and shared metrics—must be emphasized to ensure fair cross-comparison (Golmohammadi, Zhang, & Arcuri, 2023). Establishing these benchmarks and standards will accelerate progress by providing a common ground for evaluation.

Explainability and Trustworthy AI in Testing

As LLMs and ML-driven methods increasingly influence test generation, explainability becomes essential. Developers and testers need to understand why certain constraints were inferred, why specific parameter values were chosen, and why particular test sequences were prioritized. Future research should therefore integrate interpretable ML techniques—such as attention visualization, rule extraction, or post-hoc explanation methods—to increase trust in automated testing pipelines. Explainable AI in testing not only fosters adoption in industry but also helps researchers identify and mitigate failure cases (Kim, Stennett, Shah, Sinha, & Orso, 2024).

Toward Industry-Ready Testing Frameworks

Ultimately, the goal is to move beyond academic prototypes toward industry-ready RESTful API testing frameworks. This requires addressing the efficiency–accuracy trade-off, ensuring compatibility with real-world DevOps practices, and developing robust oracles. A promising direction is the design of self-adaptive testing agents that continuously monitor APIs in production environments, updating models, regenerating test cases, and detecting regressions with minimal human intervention.

Conclusion

The testing of Representational State Transfer (REST) APIs has advanced rapidly alongside their growing complexity and centrality in modern software systems. This review traces the evolution from traditional black-box and white-box methods to model-based, natural language processing (NLP), and large language model (LLM)-driven testing. Early tools such as RESTler, RestTestGen, QuickREST, and EvoMaster

established the foundation for automated testing but were limited by incomplete specifications and weak oracles. Model-driven approaches like Morest improved coverage and bug detection through execution feedback and dependency modeling.

NLP-based tools (e.g., RESTInfer, NLP2REST) enhanced specifications and input validity, while recent LLM-powered frameworks such as RESTGPT, AutoRestTest, and LlamaRestTest achieved major gains in constraint extraction, semantic accuracy, and workflow orchestration. Despite these advances, persistent challenges remain—particularly around incomplete schemas, limited semantic validation, scalability, integration of live feedback, and the performance trade-offs of large models—which continue to hinder widespread industrial adoption.

Looking ahead, RESTful API testing is moving toward hybrid frameworks that combine model-driven, heuristic, and large language model (LLM) techniques for greater semantic and structural coverage. Future work should focus on dynamic, machine learning–based oracles for richer validation, lightweight fine-tuned models for efficient CI/CD integration, and standardized benchmarks to improve comparability and reproducibility. Emphasis on explainability will also be key to building trust in ML-driven testing. Overall, machine learning, particularly LLMs and reinforcement learning, is transforming REST API testing, but progress depends on overcoming persistent challenges in specification quality, semantic validation, scalability, and efficiency to achieve reliable, industry-ready solutions.

References

Ahmed, A., & Hamdy, A. (2022). Ant bee colony for automated black box testing of RESTful API. Proceedings of the International Conference on Computer Engineering and Systems (ICCES), 72–79. https://doi.org/10.1109/ICCES56664.2022.10005116

Arcuri, A., Zhang, Z., & Galeotti, J. P. (2024). Advanced white box heuristics for search-based fuzzing of REST APIs. Proceedings of the ACM/IEEE International Conference on Software Engineering (ICSE), 114–126. https://doi.org/10.1145/3597503.3639568

Atlidakis, V., Godefroid, P., & Polishchuk, M. (2019). RESTler: Stateful REST API fuzzing. Proceedings of the 41st International Conference on Software Engineering (ICSE), 748–758. https://doi.org/10.1109/ICSE.2019.00083

Corradini, D., Zampieri, A., Pasqua, M., & Ceccato, M. (2021). RestTestGen: An extensible framework for automated black-box testing of RESTful APIs. IEEE Access, 9, 93131–93144. https://doi.org/10.1109/ACCESS.2021.3092850

Deljouyi, B., & Ramsin, R. (2022). Model-driven development of RESTful APIs. Journal of Systems and Software, 188, 111280. https://doi.org/10.1016/j.jss.2022.111280

Ehsan, A., Abuhaliqa, M. A. M. E., Catal, C., & Mishra, D. (2022). RESTful API testing methodologies: Rationale, challenges, and solution directions. Applied Sciences, 12(9), 4369. https://doi.org/10.3390/app12094369

Fertig, T., & Braun, P. (2015). Model-driven testing of RESTful APIs. Proceedings of the 24th International Conference on World Wide Web Companion (WWW '15 Companion), 1497–1502. https://doi.org/10.1145/2740908.2743045

Golmohammadi, K., Zhang, Z., & Arcuri, A. (2023). An empirical comparison of black-box test case generation tools for RESTful APIs. Empirical Software Engineering, 28(5), 1–38. https://doi.org/10.1007/s10664-023-10349-5

Karlsson, B., Čaušević, A., & Sundmark, D. (2019). QuickREST: Property-based test generation of OpenAPI-described RESTful APIs. Proceedings of the 12th IEEE Conference on Software Testing, Validation and Verification (ICST), 288–299. https://doi.org/10.1109/ICST.2019.00037

Kim, M., Stennett, T., Shah, D., Sinha, S., & Orso, A. (2024). Leveraging large language models to improve REST API testing. Proceedings of the 46th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER), 37–41. https://doi.org/10.1145/3639476.3639769

Kim, M., Sinha, S., & Orso, A. (2025). LlamaRestTest: Effective REST API testing with small language models. Proceedings of the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE). https://doi.org/10.1145/3663529.3663799

Kim, M., Stennett, T., Sinha, S., & Orso, A. (2025). AutoRestTest: A multi-agent reinforcement learning approach for REST API testing with semantic graphs and LLM-driven inputs. Proceedings of the 47th International Conference on Software Engineering (ICSE). https://doi.org/10.1145/3597503.3639501

Liu, Y., Li, Y., Deng, G., Liu, Y., Wan, R., Wu, R., Ji, D., Xu, S., & Bao, M. (2022). Morest: Model-based RESTful API testing with execution feedback. Proceedings of the 44th International Conference on Software Engineering (ICSE), 1342–1353. https://doi.org/10.1145/3510003.3510133

Liu, Y. (2022). RESTInfer: Automated inferring parameter constraints from natural language RESTful API descriptions. Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE), 112–124. https://doi.org/10.1145/3551349.3556920

Martin-Lopez, A., Segura, S., & Ruiz-Cortés, A. (2022). Enhancing REST API testing with NLP techniques. Empirical Software Engineering, 27(6), 1–30. https://doi.org/10.1007/s10664-022-10159-2

Segura, S., Martin-Lopez, A., Troya, J., & Ruiz-Cortés, A. (2018). Online testing of RESTful APIs: Promises and challenges. Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings (ICSE-C), 320–321. https://doi.org/10.1145/3183440.3194961

Song, J., Yu, T., Jiang, Z., & Chen, X. (2023). RestGPT: Connecting large language models with real-world applications via RESTful APIs. arXiv preprint, arXiv:2308.XXXX.

Yi, L., Yuekang, L., Gelei, D., Liu, Y., Wan, R., Wu, R., Ji, D., Xu, S., & Bao, M. (2022). APIRL: Deep reinforcement learning for REST API fuzzing. Proceedings of the 44th International Conference on Software Engineering (ICSE), 1453–1465. https://doi.org/10.1145/3510003.3510215

Zampieri, A., Corradini, D., Pasqua, M., & Ceccato, M. (2021). Test amplification for REST APIs using large language models. Proceedings of the 43rd International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER), 92–96. https://doi.org/10.1145/XXXX