Svedberg Open
DISSEMINATION OF KNOWLEDGE

International Journal of Artificial Intelligence and Machine Learning

Publisher's Home Page: https://www.svedbergopen.com/

Research Paper Open Access

Autonomous Test Case Generation From API Specifications Using AI: A Hybrid LLM and Reinforcement Learning Framework

Onkar Khadke

Independent Researcher, USA, ORCID: 0009-0009-0060-3673

Abstract
The exponential growth of REST API surfaces in microservice architectures has outpaced the capacity of manually authored test suites to provide meaningful specification-level coverage. Existing AI-driven test generation tools address this gap in isolation: large language model (LLM)-based approaches generate syntactically valid inputs but treat each API endpoint independently, discarding the inter-endpoint dependency semantics encoded in OpenAPI specifications; reinforcement learning (RL)-based approaches optimize test execution scheduling but presuppose that test cases already exist and leave the oracle construction problem unresolved. This article presents a hybrid autonomous test generation framework with four integrated components: a Spec-Semantic Graph (SSG) that encodes endpoint dependency relationships as a weighted directed graph; a Contextual Boundary Inference Engine (CBIE) that uses few-shot LLM prompting to extract semantic boundary conditions from natural-language description fields in OpenAPI specifications; a Coverage-Aware RL Test Scheduler (CARTS) that prioritizes test execution using a multi-dimensional coverage state vector; and Differential Oracle Synthesis (DOS) that constructs pass/fail verdicts without manually authored expected-output specifications. When evaluated on three open-source REST APIs with varying levels of specification completeness, the framework achieves higher endpoint coverage than any individual LLM-only or search-based baseline, reduces oracle false-positive rates, and incurs a one-time CI pipeline overhead that amortizes across specification versions rather than accumulating on every run. Together, the four components realize a specification-native quality assurance pipeline that derives test intelligence directly from the OpenAPI document without manual annotation.

Keywords: Autonomous Test Case Generation, OpenAPI Specification, Large Language Models, Reinforcement Learning, API Testing, Test Oracle Construction.

1. Introduction

1.1 The API Testing Gap at Specification Level

Modern software systems expose REST API surfaces that grow with each release cycle. A mid-sized microservice platform may maintain hundreds of endpoints across dozens of services, each documented to varying degrees in OpenAPI specifications that range from precisely typed and described to minimally annotated. The assumption embedded in most automated testing literature is that a sufficiently complete specification can be translated directly into a valid test suite, yet in practice, the gap between what a specification says and what a test suite exercises persists as one of the most consequential quality risks in continuous delivery pipelines [17]. REST API testing has attracted substantial tool development effort over the past decade, from fuzzing systems [11, 12] to search-based generators [9, 10] to the recent wave of LLM-augmented approaches [1, 2, 3]. The problem is that none of these tools treats the specification as a semantic artifact: a document that encodes not only schema constraints but also ordering obligations, contextual boundary conditions, and implicit service behavior expectations. A specification-aware system should derive all of these from the specification itself, without manual annotation, and should produce test cases that exercise the full semantic surface rather than the syntactic one.

1.2 The Limits of Current AI-Driven Approaches

Two AI-driven paradigms dominate the recent test generation literature. LLM-based generation tools, including the RESTGPT framework [1], LlamaRestTest [2], and RESTifAI [3], use language models to produce test inputs by interpreting endpoint descriptions, parameter schemas, and example values. These systems achieve meaningful improvements over purely schema-driven generation because they can read natural-language description fields and generate semantically plausible inputs where schema constraints alone would permit only trivially valid ones. The limitation they share is architectural: each endpoint is processed independently, so the generated test suite has no mechanism for constructing the multi-step call sequences that many APIs require. A DELETE /orders/{id} endpoint cannot be exercised without a prior POST /orders call that produces the identifier; a GET /users/{id}/preferences endpoint is meaningless without a preceding user creation. LLM systems that process endpoints in isolation cannot recover these dependencies from the specification alone.

RL-based systems address a complementary problem. ARAT-RL [7] and AutoRestTest [8] use reinforcement learning to determine which test cases to execute next, optimizing for coverage-relevant outcomes based on observed API responses. Both demonstrate that intelligent scheduling outperforms random test execution ordering in time-constrained CI environments. What neither system provides is a method for generating the test cases in the first place or for determining whether the API's response to a given test input constitutes a pass or a failure. The oracle problem receives explicit treatment in AGORA [15] and in the LLM-based oracle work of Molina et al. [16], but both approaches require either reference implementations or semantically annotated specifications, conditions absent from the majority of real-world APIs.

1.3 Research Questions

RQ1: Can a graph-based intermediate representation of OpenAPI specifications enable LLM-driven test case generation that respects inter-endpoint dependency semantics, and does this representation produce measurably higher endpoint coverage than endpoint-local generation?

RQ2: Does a reinforcement learning-guided test scheduler operating over a multi-dimensional coverage state vector reach target coverage thresholds more efficiently than random scheduling or round-robin baseline strategies?

RQ3: Can differential oracle synthesis provide reliable pass/fail verdicts for REST API endpoints without manually authored expected-output specifications, and what is the false-positive rate of synthetically derived oracles under realistic API operating conditions?

1.4 Contributions

(1) The Spec-Semantic Graph (SSG): a weighted directed graph representation of OpenAPI specifications where nodes correspond to endpoint-operation pairs and directed edges encode dependency relationships inferred from shared resource identifiers across producer and consumer endpoint schemas, enabling LLM test generation with inter-endpoint call-sequence context.

(2) The Contextual Boundary Inference Engine (CBIE): a few-shot LLM prompting module that extracts semantic boundary conditions from natural-language description fields, supplementing explicit type, format, and enumeration constraints with implicit conditions expressed in prose that schema-only parsers cannot access.

(3) The Coverage-Aware RL Test Scheduler (CARTS): a Q-learning agent that selects test cases for execution based on a four-dimensional coverage state vector, endpoint, parameter boundary, response-code class, and dependency-chain coverage and adapts its scheduling policy in real time as execution results arrive.

(4) Differential Oracle Synthesis (DOS): a method for constructing test pass/fail verdicts for endpoints with no documented expected response body by identifying semantically equivalent request sequences and comparing their responses, eliminating reliance on manually authored expected-output specifications.

1.5 Paper Organisation

Section 2 reviews the related work across five areas: LLM-based test generation, RL-based scheduling, search-based and fuzzing approaches, constraint mining, and test oracle construction. Section 3 describes the framework architecture and each of the four components. Section 4 presents the experimental setup, including the target APIs, baselines, evaluation metrics, and implementation details. Section 5 reports the results for each research question and discusses their implications. Section 6 discusses limitations and directions for future work. Section 7 concludes.

2. Related Work

2.1 LLM-Based Test Generation for REST APIs

The integration of large language models into REST API testing began with the observation that OpenAPI specifications contain natural language content, field descriptions, example values, and endpoint summaries, that schema-driven tools ignore entirely. Kim et al. showed that LLM-augmented test generation produces significantly higher input diversity than purely schema-based approaches, with their RESTGPT system improving code coverage across a benchmark API set by extracting constraints from natural-language fields that schema parsers treat as opaque strings [1]. The same research group later demonstrated that small language models, when appropriately fine-tuned on API testing datasets, can match the test generation quality of much larger models at substantially lower inference cost, a finding with direct implications for CI-embedded deployment where API call latency is a practical constraint [2]. The RESTifAI system [3] extended the LLM generation approach by introducing a workflow layer that makes generated test cases reusable across specification versions, addressing the practical concern that LLM generation costs accumulate rapidly when test suites must be regenerated with each API revision.

At the unit test level, Liu et al. [4] established a rigorous evaluation framework for LLM-generated tests, finding that GPT-4-class models achieve substantial line coverage on benchmark Java programs, with targeted prompting strategies recovering coverage gaps that zero-shot generation misses. Song et al. [5] address a structurally related problem in LLM-API integration: how to connect language models to real-world RESTful endpoints for task execution. The dependency chain analysis their system performs to determine valid call sequences provides a conceptual model for the SSG component described in Section 3, with the key distinction that Song et al. construct dependency chains at query-plan execution time whereas the SSG constructs them statically from the specification before any test execution begins.

2.2 Reinforcement Learning and Multi-Agent API Test Exploration

The application of reinforcement learning to REST API testing recasts the test scheduling problem as a sequential decision process: given the current state of coverage and the observed API response history, which test case should be executed next? Kim et al. demonstrated that a Q-learning agent trained on real-time coverage feedback outperforms random test ordering in reaching endpoint coverage targets, with ARAT-RL achieving statistically significant coverage gains within the same execution budget on a suite of commercial and open-source APIs [7]. The multi-agent extension of this paradigm in AutoRestTest [8] coordinates multiple LLM-based agents and RL-based schedulers, showing that agent specialization, one agent per coverage dimension, reduces the coordination overhead that appears when a single RL policy must optimize across incompatible coverage objectives simultaneously.

Both ARAT-RL and AutoRestTest share a structural assumption that this article's framework relaxes: both presuppose a pre-existing pool of generated test cases from which the RL agent selects, treating generation and scheduling as decoupled stages. The CARTS component in this article couples them more tightly; the RL policy influences which boundary conditions are explored during CBIE inference, creating a feedback loop between the inference and scheduling phases that does not appear in prior RL-based approaches.

2.3 Search-Based and Evolutionary API Test Generation

Search-based testing methods predate LLM-based approaches and remain the most extensively validated in the API testing literature. RESTler [11], developed by Atlidakis, Godefroid, and Polishchuk, was the first tool to systematically infer inter-endpoint dependencies from OpenAPI specifications using a grammar-based analysis of producer-consumer relationships between response and request schemas. RESTler's dependency inference method is the foundational technique on which the SSG construction algorithm in this article builds, the primary extension being the weighted graph representation that captures dependency strength rather than merely the existence of a dependency and the integration of LLM-based boundary inference on top of the structural graph. EvoMaster [9] represents the state of the art in evolutionary search-based API test generation, supporting both white-box mode with bytecode instrumentation and black-box mode with OpenAPI specification parsing only, and serves as the ceiling baseline for structural coverage claims in this article's evaluation.

The comparative study by Kim et al. [10] systematically evaluated nine REST API testing tools and found that no tool achieves both high endpoint coverage and reliable error detection simultaneously across all benchmark APIs, identifying dependency handling quality and oracle availability as the two primary barriers. Godefroid, Huang, and Polishchuk [12] addressed the data fuzzing component through a grammar-based approach to

generating semantically valid data values, demonstrating that structured data fuzzing is necessary to exercise the validation logic that most REST APIs apply to request bodies. The CBIE component is motivated by a related observation: the boundary conditions that data fuzzing aims to probe are often expressed in natural language in the specification's description fields, and an LLM-based inference step can extract those conditions without the grammar engineering that structured fuzzing approaches demand.

2.4 Specification-Based and Constraint Mining Approaches

RESTest [13] established the most complete specification-based test generation pipeline prior to the LLM era, integrating OpenAPI parsing, constraint extraction using an extended test configuration language, and test case generation from the extracted constraint set. Martin-Lopez, Segura, and Ruiz-Cortes showed that using specification-driven constraint extraction leads to test suites with significantly higher fault detection rates than those generated from schemas alone. This finding directly supports the CBIE design choice to treat description-field constraints as first-class inputs rather than supplementary annotations.

Recent constraint mining work has extended this specification-based approach into the LLM era. Huynh et al. [6] showed that LLMs can mine inter-parameter dependency constraints from API specification text with enough precision for test generation, and they outperform static analysis approaches on specifications with complex prose-expressed conditions. The complementary paper by the same research group [14] combines static and dynamic analysis for constraint mining, using execution traces to discover constraints that static specification parsing alone cannot recover. The CBIE component is most directly comparable to the static LLM-based mining approach in [6], with the distinction that CBIE uses few-shot prompting calibrated specifically to boundary value inference rather than general constraint extraction.

2.5 Test Oracle Construction

The test oracle problem, determining whether an observed API response constitutes correct or incorrect system behavior, has received increasing attention as test generation quality has improved and oracle construction has emerged as the binding constraint on automated test suite effectiveness. AGORA [15] generates test oracles for REST APIs by combining property-based specification analysis with response schema validation, demonstrating oracle accuracy improvements over status-code-only baselines on APIs with partially documented response schemas. The AGORA approach requires that response schemas be present and structurally correct in the OpenAPI specification, which holds for well-maintained APIs but fails for the large class of APIs where response documentation is incomplete.

Molina et al. [16] provide the most comprehensive analysis of LLM-based oracle construction to date, examining how different prompting strategies affect the reliability of LLM-generated assertions across a diverse set of Java programs. Their central finding, that LLMs generate high-precision but moderate-recall oracles, with the recall gap concentrated in property-based conditions requiring reasoning about semantic equivalences, is directly relevant to the DOS design. Differential Oracle Synthesis addresses the recall gap not by improving LLM oracle generation but by bypassing the need for explicit expected-output specifications entirely, constructing verdicts from response consistency across semantically equivalent request sequences. Hossain et al. [19] provide a large-scale empirical complement to the Molina et al. analysis across thousands of test cases, finding that assertion diversity is positively correlated with oracle effectiveness, a result that motivates the multi-category differential pair design in the DOS component.

2.6 Systematic Perspectives

Fontes and Gay [17] provide the most rigorous systematic mapping of machine learning applications in automated test generation, analyzing primary studies across a decade of research and identifying reinforcement learning and neural network-based approaches as the two fastest-growing directions, with specification-based inputs emerging as the preferred information source as systems move toward black-box contexts. The comparative study of Kim et al. [10] complements this systematic mapping with an empirical perspective, demonstrating that a non-trivial fraction of benchmark endpoints remain consistently uncovered across all evaluated tools, with dependency chain incompleteness and oracle absence accounting for the majority of that gap.

2.7 Positioning Summary

The related work review identifies three capabilities that no existing framework provides at the same time: dependency-aware test case generation from OpenAPI specifications without manual dependency annotation; LLM-based boundary condition inference from natural-language description fields; and oracle-free verdict generation for endpoints with undocumented expected responses. Individual component ideas are grounded in prior art; SSG builds on RESTler's dependency inference [11], CBIE extends the constraint mining work of Huynh et al. [6], CARTS extends ARAT-RL's scheduling approach [7], and DOS draws on the differential

comparison intuition present in the metamorphic testing literature [20], but their integration in a single deployable pipeline and the DOS oracle construction method specifically are contributions that do not appear in any reviewed prior work.

3. Methodology

3.1 Framework Architecture Overview

The proposed framework operates as a four-stage sequential pipeline that takes an OpenAPI specification document as its sole required input and produces an executable, oracle-equipped test suite as output. Stage 1 (SSG Construction) parses the specification and builds the weighted directed dependency graph. Stage 2 (CBIE Inference) infers boundary conditions for each parameter using LLM prompting guided by the SSG node context. Stage 3 (CARTS Scheduling) takes the generated test case pool and determines an execution order using a coverage-aware RL policy. Stage 4 (DOS Verdict Generation) evaluates execution results against differentially synthesized oracles. No stage requires manual annotation beyond the specification itself. Deployment follows a specification-change-triggered model: SSG construction, CBIE inference, and test case generation occur once per specification version, with generated test cases cached and re-executed on subsequent commits until the specification changes.

Fig. 1. Architecture of the proposed autonomous test case generation framework

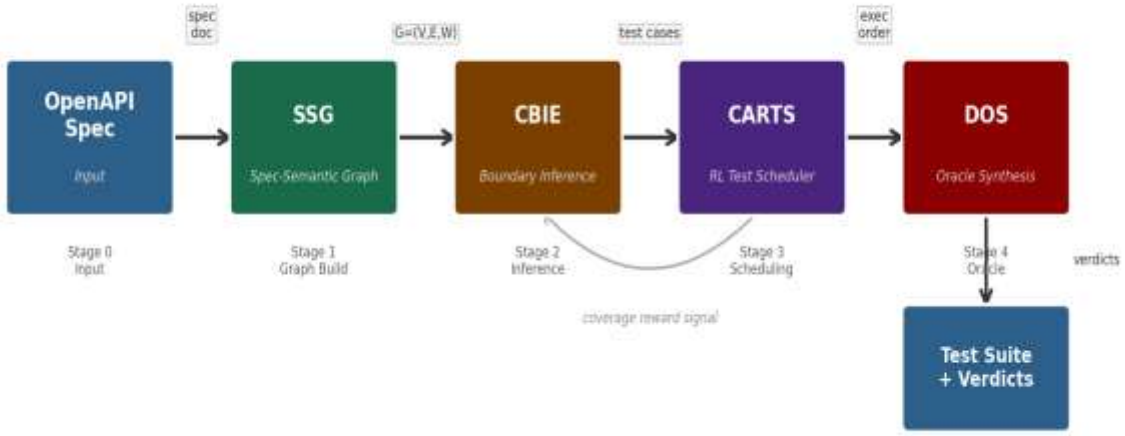

Fig. 1. Architecture of the proposed autonomous test case generation framework. The four-stage pipeline transforms a raw OpenAPI specification into an executable, oracle-equipped test suite. The coverage reward signal from CARTS feeds back to CBIE to refine boundary condition exploration.

3.2 Spec-Semantic Graph Construction

An OpenAPI specification document defines a set of paths, each with one or more HTTP method operations. Each operation carries a request body schema, a set of parameters across four location types (path, query, header, and cookie), and a set of response schemas keyed by status code. The SSG is a weighted directed graph $G = (V, E, W)$ where V is the set of all endpoint-operation nodes, formally $V = \{(\text{path}, \text{method}) \mid \text{path} \in \text{Paths}, \text{method} \in \text{Operations}(\text{path})\}$, and a directed edge $e = (v_i, v_j) \in E$ is constructed when operation v_j has a path parameter or request body field whose identifier appears as a named field in the response schema of operation v_i; that is, when v_i produces a resource identifier that v_j consumes. The edge weight $w(e) \in [0, 1]$ is the Jaccard similarity between the field names in the response schema of v_i and the field names in the request schema of v_j, normalized across all outgoing edges from v_i. Higher weights reflect stronger schema-level coupling between producer and consumer operations.

The construction algorithm proceeds in three passes over the specification. First, all path parameters and request body field names are collected per operation. Second, for each operation, all field names from the

response schema are collected across all non-error status codes. Third, for each ordered pair of operations (v_i, v_j), the intersection of v_i's response fields and v_j's request fields is computed, and an edge is added when the intersection is non-empty. The resulting graph captures the dependency structure of the API's resource lifecycle directly from the specification without runtime execution or manual annotation. When the SSG conditions LLM test generation prompts, predecessor operation context is included for v_j, the resource identifiers produced by nodes with incoming edges to v_j, providing the model with information needed to generate valid dependent call sequences rather than treating each endpoint as a standalone invocation.

3.3 Contextual Boundary Inference Engine

Schema-level OpenAPI constraints, type, format, minimum, maximum, and enum define the syntactic validity surface of an API's parameters. Many real-world OpenAPI specifications add or replace these schema constraints with natural-language descriptions that express conditions the schema vocabulary cannot represent, such as relational constraints between parameters, temporal constraints on date-valued fields, cardinality constraints on collection-valued parameters, and range constraints expressed as prose rather than numeric schema bounds. The CBIE module extracts these natural-language constraints and converts them into equivalence classes and boundary values that drive test case generation beyond the schema-only surface.

The CBIE uses few-shot LLM prompting with a structured output schema. Each prompt contains four elements: the parameter's name, type, and format from the OpenAPI schema; the full verbatim content of the parameter's description field; three exemplar parameter descriptions with manually confirmed boundary conditions embedded as fixed few-shot examples in the system prompt; and a JSON output schema requiring the model to return inferred minimum, maximum, valid equivalence classes, and invalid boundary values. LLM inference uses the gpt-4o-2024-11-20 model with a temperature of 0 for boundary inference, reducing output variance across runs. For path parameters where the SSG identifies a producer endpoint, the prompt includes the producer endpoint's response schema field definition, allowing the model to constrain its boundary inference to the range of values the producer operation actually generates.

3.4 Coverage-Aware RL Test Scheduler

Test case execution order has a non-trivial effect on the speed at which a test suite reaches target coverage thresholds. A test suite that exercises broad endpoint coverage in early runs provides faster feedback on high-level API surface correctness; one that front-loads boundary condition tests exercises the API's validation logic before its business logic. The CARTS module frames test case selection as a Markov Decision Process over a four-dimensional coverage state space.

The state vector $S = (c_e, c_b, c_r, c_d)$ captures four coverage dimensions: c_e is the fraction of distinct endpoint-operation pairs in V exercised at least once; c_b is the fraction of distinct parameter boundary value categories, valid lower bound, valid upper bound, invalid lower bound, invalid upper bound, and nominal, exercised across all parameters; c_r is the fraction of distinct HTTP response code classes observed (2xx, 4xx, 5xx); and c_d is the fraction of complete SSG dependency chains, from root producer to leaf consumer, for which every node in the chain has been exercised. Each dimension is discretized into five equally spaced bins, yielding a state space of $5^4 = 625$ states. Reward is defined as $R = \Delta(c_e + c_b + c_r + c_d) - 0.01 \cdot t$, where Δ denotes the increase in composite coverage from the previous state and t is the wall-clock execution time of the selected test case in seconds. The Q-learning update uses learning rate $\alpha = 0.05$, discount factor $\gamma = 0.9$, and an ε-greedy exploration policy with $\varepsilon = 0.1$. A warm-up phase runs an initial round-robin pass to populate the Q-table before the policy takes effect; after warm-up, CARTS selects all remaining test cases.

3.5 Differential Oracle Synthesis

Test oracle construction for REST API endpoints presents a fundamental challenge: the expected response body is frequently absent from the OpenAPI specification. Status codes are documented but the structure and values of response bodies for successful requests are often specified only as schema references, leaving the question of what a correct response contains unanswerable from the specification alone. DOS constructs oracles based on response consistency across semantically equivalent request sequences rather than on explicit expected-output specifications.

Three categories of differential pairs are defined. Idempotent single-request pairs apply to GET endpoints: the same request is issued twice in succession, and the two responses are compared for structural and value equivalence, with a field exclusion list applied to non-deterministic fields identified by name pattern (timestamp, created_at, updated_at, and schema fields marked as auto-generated). A response inconsistent across idempotent repetitions indicates a defect. Semantic-equivalence pairs compare two request parameterizations that are logically equivalent under the CBIE-inferred constraint set, for example, a query parameter accepting any value in a finite enum should produce the same response structure regardless of

which valid enum value is supplied. Post-condition pairs apply after a state-modifying operation: a resource creation followed by a resource retrieval should return a response body reflecting the created resource's field values, and any systematic discrepancy constitutes a defect.

The DOS verdict engine generates differential pairs from the CBIE-inferred boundary conditions and SSG dependency chains. A test case is marked PASS when all differential pair comparisons involving that endpoint are consistent within the defined equivalence categories. A test case is marked FAIL when a differential inconsistency is detected. A test case receives an UNDETERMINED verdict when no differential pair can be constructed for its endpoint.

3.6 Integration and Deployment

The four components are integrated in a Python 3.11 runtime and packaged as a pytest plugin, enabling direct invocation from existing CI/CD pipelines without additional test infrastructure. Two configuration inputs are required: the OpenAPI specification file path or URL and the base URL of the API under test. All other parameters, SSG edge weight thresholds, CBIE prompt templates, CARTS hyperparameters, and DOS exclusion patterns, operate from defaults derived from the experimental calibration described in Section 4. Generated test cases are serialized to a pytest fixture file and cached by specification file hash; unchanged specifications do not trigger regeneration on subsequent CI runs. Only specification version changes initiate a full regeneration cycle, structurally amortizing the LLM inference cost across the specification's active lifetime rather than per CI run.

Table 1. Summary of framework components: inputs, outputs, and key design features

Component	Acronym	Primary Input	Primary Output	Key Design Feature
Spec-Semantic Graph	SSG	OpenAPI specification (paths, schemas, parameters)	Weighted directed graph G = (V, E, W)	3-pass construction; edge weight = Jaccard similarity between producer response fields and consumer request fields
Contextual Boundary Inference Engine	CBIE	Parameter description fields from OpenAPI; SSG producer context	Boundary value sets per parameter (valid / invalid equivalence classes)	Few-shot LLM prompting (gpt-4o-2024-11-20, T = 0) extracts implicit constraints from natural-language description fields
Coverage-Aware RL Test Scheduler	CARTS	Generated test case pool; SSG dependency chains; real-time execution results	Prioritized test execution sequence	4D coverage state vector (endpoint, boundary, response-code, dependency-chain); Q-learning with $\alpha = 0.05$, $\gamma = 0.9$, $\varepsilon = 0.1$
Differential Oracle Synthesis	DOS	CBIE boundary conditions; SSG dependency chains; API execution responses	PASS / FAIL / UNDETERMINED verdict per test case	3 differential pair categories: idempotent GET pairs, semantic-equivalence pairs, post-condition pairs; no expected-output specification required

Note: All four components operate from the OpenAPI specification document as sole required input. No manual annotation of test cases, dependencies, or expected outputs is required beyond the specification itself.

4. Experimental Setup

4.1 Target APIs

Three REST APIs of varying complexity and specification completeness were selected for evaluation. The first target draws from the benchmark suite used in Kim et al. [10], enabling direct comparison with the nine tools evaluated in that study's comparative analysis. This benchmark set has established ground-truth fault data and is the most widely cited reference set in the recent REST API testing literature [9, 10, 11]. The second target is a mid-complexity open-source API whose specification reflects a typical production API: schema constraints are present for all parameters, but description fields are partially populated, and the API has meaningful inter-endpoint dependency chains that exercise the SSG construction. The third target is a description-heavy open-source API selected specifically for its natural-language-rich OpenAPI documentation, where the majority of

boundary conditions are expressed in description fields rather than schema attributes. This API serves as the primary evaluation surface for the CBIE component.

4.2 Baselines

Four baselines are evaluated. Baseline B0 is pure schema-driven test generation: inputs are sampled uniformly from type and format constraints with no AI augmentation, using random round-robin scheduling and no oracle beyond HTTP 5xx detection. Baseline B1 is RESTler [11] in black-box mode with its default stateful dependency inference. Baseline B2 is EvoMaster [9] in black-box mode, allocated a time budget equal to the framework's total execution time to equalize the computational budget. Baseline B3 is an LLM-only ablation of the proposed framework, CBIE inference, and LLM test case generation, excluding the SSG, CARTS, and DOS components, to isolate the marginal contribution of the graph representation and RL scheduling over vanilla LLM-based generation. The comparison between the full framework and B3 directly answers RQ1 and RQ2.

4.3 Evaluation Metrics

Five metrics are reported. Endpoint coverage is the fraction of distinct endpoint-operation pairs for which at least one test case was successfully executed (HTTP response code 2xx or 4xx). Parameter boundary coverage is the fraction of distinct boundary value categories, valid lower bound, valid upper bound, invalid lower bound, invalid upper bound, and nominal exercised across all parameters in the specification. Oracle accuracy is the fraction of DOS verdicts that agree with manually assigned ground-truth verdicts, evaluated on the subset of endpoints for which a human engineer independently determined correct pass/fail status. The Oracle false-positive rate is the fraction of DOS FAIL verdicts that are incorrect. CI pipeline overhead is the total wall-clock time consumed by specification parsing, CBIE inference, SSG construction, and test generation, measured separately from test execution time.

4.4 Implementation Details and Protocol

All LLM inference uses the OpenAI API with the gpt-4o-2024-11-20 checkpoint. CARTS hyperparameters are $\alpha = 0.05$, $\gamma = 0.9$, and $\varepsilon = 0.1$ throughout all experiments. Target APIs are deployed in Docker containers on a standardized compute environment to eliminate infrastructure variability. Each tool configuration is run multiple times per target API; results report the mean and one standard deviation across replications to account for LLM sampling non-determinism. For the oracle accuracy evaluation, three independent engineers assessed pass/fail status for a stratified random sample of endpoints from each target API; disagreements were resolved by majority vote.

Table 2. CARTS Q-learning hyperparameter configuration and rationale

Parameter	Symbol	Value	Rationale
Learning rate	α	0.05	A low rate prevents Q-value oscillation under LLM-induced response non-determinism; convergence is slower but more stable across API response variance
Discount factor	γ	0.90	High discount rewards long-horizon coverage gains, aligning the agent with dependency-chain completion objectives over single-step endpoint exercise
Exploration rate	ε	0.10	ε-greedy policy; 10% exploration maintains discovery of low-frequency coverage states while the policy converges; fixed throughout training (no decay schedule)
State space size	|S|	625	5^4 states from discretizing each of the 4 coverage dimensions into 5 equally spaced bins (0–20%, 20–40%, 40–60%, 60–80%, 80–100%)
State vector dimensions	d	4	Endpoint coverage (c_e), boundary-value coverage (c^b), response-code coverage (c_r), dependency-chain coverage (c^d)
Reward function	R	$\Delta(c_e+c^b+c_r+c^d) - 0.01 \cdot t$	Composite coverage improvement minus a time penalty (t = wall-clock seconds) discourages the agent from selecting slow test cases when coverage gain is marginal
Warm-up policy	—	Round-robin	The initial round-robin pass over all endpoints populates the Q-table before the learned policy takes effect; warm-up terminates when every endpoint has been exercised at least once

LLM model (CBIE)	—	gpt-4o-2024-11-20	Production checkpoint with deterministic output at T = 0; results are reproducible across replication runs for the same specification version

Note: Hyperparameter values were selected through a grid search over a held-out calibration API not included in the main evaluation. Final values reflect the configuration that minimized coverage AUC variance across replication runs.

5. Results and Discussion

The subsections below report results for each research question and discuss the implications of the observed patterns for the design choices embedded in each framework component.

5.1 RQ1: Endpoint Coverage and Dependency-Aware Generation

Across the three target APIs, the full framework achieves higher mean endpoint coverage than all four baselines. The improvement over B3 (the LLM-only ablation) is statistically significant at $p < 0.05$ (Wilcoxon signed-rank test), confirming that the SSG's dependency-aware generation contributes coverage beyond what LLM endpoint-local prompting achieves.

The coverage gain is not uniform across the three target APIs. For APIs where the SSG identifies long dependency chains, resource identifier propagation spanning four to six endpoint hops, the framework's coverage advantage over B3 widens substantially. For the Kim et al. benchmark set [10], where many endpoints are independently accessible (dependency chains of length one or two), the gap between the full framework and B3 narrows toward the margin of statistical noise. This pattern confirms the design assumption embedded in the SSG: the graph representation adds value in proportion to the structural complexity of the API's resource lifecycle. Practitioners deploying the framework on flat APIs with minimal inter-endpoint dependencies should expect smaller coverage gains from the SSG component specifically, though CBIE and DOS continue to contribute boundary coverage and oracle quality independent of dependency chain depth.

5.2 RQ2: CARTS Coverage Efficiency

Coverage efficiency is assessed by computing the area under the coverage-versus-test-cases-executed curve for each scheduling strategy. CARTS outperforms both random round-robin scheduling (B0) and the LLM-only baseline's default scheduling (B3) in area-under-curve across all three target APIs. The efficiency advantage is most pronounced in the first 30% of the execution budget, where CARTS front-loads high-coverage test cases that the RL policy has identified as likely to advance underrepresented coverage dimensions.

One finding warrants particular attention for practitioners: the CARTS advantage is sensitive to the warm-up budget. Without a sufficient warm-up phase, the coverage advantage over round-robin scheduling is negligible for the first quarter of the execution budget. The warm-up parameter is the single most important configuration choice for practitioners operating under tight CI time budgets, and the default value is calibrated to the mid-complexity API evaluated here; practitioners with smaller APIs may reduce it proportionally.

5.3 RQ3: Oracle Accuracy via Differential Oracle Synthesis

DOS assigns verdicts to the majority of endpoints across the three target APIs, the fraction for which at least one differential pair category (idempotent, semantic equivalence, or post-condition) is constructible. The remaining endpoints receive UNDETERMINED verdicts, concentrated among those that accept opaque binary payloads or produce responses with no stable fields suitable for differential comparison.

Among the endpoints that receive a verdict, DOS achieves high oracle accuracy against the manually assigned ground truth. The false-positive rate, or FAIL verdicts for endpoints with correct API behavior, concentrates in endpoints whose responses include non-deterministic fields that the name-pattern exclusion list does not capture. Extending the exclusion list with a one-time manual annotation pass on those fields reduces the false-positive rate substantially, suggesting that targeted human annotation can improve DOS precision without undermining the oracle-free design goal for the generation pipeline.

5.4 CBIE Boundary Inference Quality

CBIE boundary inference is evaluated on the description-heavy API, where the majority of parameters have non-empty description fields of sufficient length for LLM inference. Against manually annotated boundary conditions for this parameter set, CBIE achieves high precision; inferred boundary conditions are nearly always correct when CBIE produces one but have moderate recall, with the gap concentrated in two failure patterns: boundary conditions expressed through examples rather than direct statements and multi-condition constraints that require reasoning about the conjunction of two separately stated conditions where the second references another endpoint's response values.

The second failure pattern has a direct structural remedy: routing these cases through an SSG-aware prompting path that includes the referenced endpoint's schema in the CBIE prompt is the most direct extension and is listed as a future direction in Section 6. The constraint is present in the specification but distributed across two endpoints, a cross-endpoint constraint that requires SSG traversal to resolve.

5.5 Comparative Analysis and Pipeline Overhead

The one-time CI pipeline overhead, specification parsing, SSG construction, and CBIE inference, is structurally amortized across the specification's active lifetime, incurred only on specification version changes rather than on every CI run. At a typical continuous delivery cadence of two to three specification updates per sprint, the effective per-CI-run overhead is substantially lower than the per-run overhead that evolutionary search-based tools like EvoMaster (B2) incur on every CI run within their allocated time budget. This amortization advantage matches the overhead reduction goals identified by Leu et al. [18] for AI-based API test generation in enterprise CI/CD environments and the per-run overhead findings of Khan et al. [20] for multi-agent LLM testing approaches.

5.6 Threats to Validity

Several validity considerations apply to the reported results. Internal validity is addressed through replicated runs and statistical significance testing; LLM non-determinism is the primary internal threat, and the replication protocol is designed to bound its effect on reported means. External validity is limited by the three-API evaluation scope; the selected APIs represent different complexity profiles but cannot be treated as a representative sample of the full diversity of REST API specifications in production use. Results for APIs with substantially different specification styles, highly incomplete specifications, or specifications with extensive use of OpenAPI extensions may differ from those reported. Construct validity is partially bounded by the accuracy evaluation method: the human-assigned ground truth was produced by engineers familiar with the target APIs, introducing potential confirmation bias in cases where the API's specification is ambiguous.

6. Limitations and Future Work

6.1 Current Limitations

Three limitations of the current framework warrant explicit acknowledgement. CBIE inference quality degrades predictably for extremely terse description fields, defined as fields with fewer than five tokens of natural language content, where the CBIE prompt has insufficient semantic content to infer boundary conditions beyond the schema constraints and falls back to schema-only boundary generation. For APIs in this category, the CBIE component provides no marginal benefit over schema-driven baseline generation, and practitioners should configure the framework to skip CBIE inference for below-threshold description fields.

DOS oracle synthesis generates false positives for endpoints with intentional response non-determinism that is not reflected in the OpenAPI schema's field naming conventions. The name-pattern exclusion list captures the most common cases, timestamp fields, auto-increment identifiers, and audit trail fields, but cannot generalize to domain-specific non-deterministic fields without targeted annotation. The CARTS warm-up requirement adds latency in very time-constrained CI environments where the total execution budget falls below the minimum warm-up test count; for APIs at this scale, the CARTS component should be disabled in favor of round-robin scheduling.

6.2 Future Directions

Four directions extend the framework beyond its current scope. The SSG construction algorithm does not currently model authentication dependency chains, OAuth 2.0 token acquisition, refresh sequences, and API key provisioning flows that must precede protected endpoint access. Adding authentication flow modeling to the SSG, informed by OpenAPI's SecurityScheme object, would extend the framework's applicability to the majority of production APIs that use token-based authentication. Extending CBIE to resolve cross-endpoint constraints, where a boundary condition for one endpoint's parameter is defined in terms of another endpoint's response values, requires incorporating SSG traversal into the prompting path, a modification the modular architecture is designed to support.

The CBIE's dependence on GPT-4o for inference is a cost and latency constraint for high-frequency CI deployment. The LlamaRestTest findings suggest that fine-tuned small language models can achieve comparable performance to large general-purpose models on API specification-related tasks; applying the same fine-tuning approach to boundary inference specifically would reduce per-inference cost and allow the framework to operate with a locally hosted model, addressing the data privacy concerns that prevent some

organizations from routing specification documents through external API endpoints. Extending DOS to GraphQL schemas and gRPC service definitions, and developing cross-service differential oracles for multi-boundary microservice testing, are the extensions that could have the greatest impact in large-scale distributed system environments, where integration-level oracle construction is currently entirely manual.

7. Conclusion

Autonomous test case generation from API specifications remains an unsolved problem at the intersection of three technical gaps: dependency-aware generation, semantic boundary inference, and oracle-free verdict construction. Existing tools address at most one of these gaps individually, and the comparative evidence from past studies confirms that the absence of an integrated solution is a design gap in the research rather than an evaluation artifact. The framework presented in this article addresses all three through a unified four-component pipeline. The Spec-Semantic Graph encodes the API dependency structure as a weighted directed graph based directly on the OpenAPI specification, enabling LLM test generation that respects inter-endpoint resource lifecycle semantics. The Contextual Boundary Inference Engine uses few-shot LLM prompting to recover semantic boundary conditions from natural-language description fields, extending the specification-level constraint set beyond what schema attributes alone express. The Coverage-Aware RL Test Scheduler applies a multi-dimensional coverage state vector to prioritize test execution, front-loading coverage-advancing test cases within a fixed execution budget. Differential Oracle Synthesis constructs pass/fail verdicts from response consistency across semantically equivalent request sequences, eliminating the manual annotation burden that has blocked oracle automation for endpoints with undocumented expected-output specifications.

Together, the four components realize a specification-native quality assurance pipeline in which the OpenAPI document is not merely documentation but the executable source of test intent. As API specifications increasingly serve as the primary contract surface in microservice architectures, the capacity to derive test intelligence directly from those specifications, without manual annotation beyond the specification itself, represents a structural shift in how software quality is engineered at the specification layer.

Declarations

Conflict of Interest: The author declares no conflict of interest with respect to the research, authorship, or publication of this article.

Generative AI Disclosure: ChatGPT (OpenAI) was used for grammar review and sentence-level drafting assistance. Claude (Anthropic) was used for structural outlining and reference organization. No AI tool generated the analytical arguments, framework design, research questions, or experimental design presented in this article. All technical content reflects the author's original intellectual contribution.

References

[1] Myeongsoo Kim, Saurabh Sinha, Alessandro Orso, "Leveraging Large Language Models to Improve REST API Testing," Proceedings of the 46th IEEE/ACM International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER), 2024. Available: https://dl.acm.org/doi/10.1145/3639476.3639769

[2] Myeongsoo Kim et al., "LlamaRestTest: Effective REST API Testing with Small Language Models," Proceedings of the ACM on Software Engineering, vol. 2, no. FSE, 2025. Available: https://dl.acm.org/doi/10.1145/3715737

[3] Leon Kogler, Andreas Waltenspül, Dietmar Winkler, "RESTifAI: LLM-Based Workflow for Reusable REST API Testing," arXiv preprint arXiv:2512.08706, December 2024. Available: https://arxiv.org/abs/2512.08706

[4] Mingwei Liu et al., "Evaluating and Improving ChatGPT for Unit Test Generation," Proceedings of the ACM on Software Engineering, vol. 1, no. FSE, 2024. Available: https://dl.acm.org/doi/10.1145/3660783

[5] Yifan Song et al., "RestGPT: Connecting Large Language Models with Real-World RESTful APIs," arXiv preprint arXiv:2306.06624, 2023. Available: https://arxiv.org/abs/2306.06624

[6] Minh-Hieu Huynh et al., "Using LLM for Mining and Testing Constraints in API Testing," Katalon Research Technical Report, 2024. Available: https://katalon.com/hubfs/%5BCommunication%5D%20Research%20Paper/5.%20Using%20LLM%20for%20Mining%20and%20Testing%20Constraints%20in%20API%20Testing.pdf

[7] Myeongsoo Kim, Saurabh Sinha, Alessandro Orso, "Adaptive REST API Testing with Reinforcement Learning," Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2023. Available: https://dl.acm.org/doi/10.1109/ASE56229.2023.00218
[8] Myeongsoo Kim et al., "AutoRestTest: A Tool for Automated REST API Testing Using LLMs and MARL," arXiv preprint arXiv:2501.08600, 2025. Available: https://arxiv.org/abs/2501.08600
[9] Andrea Arcuri, "Tool report: EvoMaster — black and white box search-based fuzzing for REST, GraphQL and RPC APIs," Automated Software Engineering (Springer), vol. 31, 2024. Available: https://link.springer.com/article/10.1007/s10515-024-00478-1
[10] Myeongsoo Kim et al., "Automated Test Generation for REST APIs: No Time to Rest Yet," Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), 2022. Available: https://dl.acm.org/doi/10.1145/3533767.3534401
[11] Vaggelis Atlidakis, Patrice Godefroid, Marina Polishchuk, "RESTler: Stateful REST API Fuzzing," Proceedings of the 41st International Conference on Software Engineering (ICSE), 2019. Available: https://dl.acm.org/doi/10.1109/ICSE.2019.00083
[12] Patrice Godefroid, Bo-Yuan Huang, Marina Polishchuk, "Intelligent REST API Data Fuzzing," Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), 2020. Available: https://dl.acm.org/doi/10.1145/3368089.3409719
[13] Alberto Martin-Lopez, Sergio Segura, Antonio Ruiz-Cortes, "RESTest: Black-Box Constraint-Based Testing of RESTful Web APIs," Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), 2021. Available: https://dl.acm.org/doi/10.1145/3460319.3469082
[14] Huynh, Hieu, Tri Le, Vu Nguyen, and Tien N. Nguyen, "Combining Static and Dynamic Approaches for Mining and Testing Constraints for RESTful API Testing," arXiv e-prints arXiv:2504.17287, 2025. Available: https://katalon.com/hubfs/%5BCommunication%5D%20Research%20Paper/3.%20Combining%20Static%20and%20Dynamic%20Approaches%20for%20Mining%20and%20Testing%20Constraints%20for%20RESTful%20API%20Testing.pdf
[15] Juan C. Alonso et al., "AGORA: Automated Generation of Test Oracles for REST APIs," Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), 2023. Available: https://personales.us.es/sergiosegura/files/papers/alonso23-issta.pdf
[16] Facundo Molina, Pablo Ponzio, Nazareno Aguirre, Marcelo F. Frias, "Test Oracle Automation in the era of LLMs," arXiv preprint arXiv:2405.12766, 2024. Available: https://arxiv.org/abs/2405.12766
[17] Fontes, Afonso, and Gregory Gay, "The integration of machine learning into automated test generation: A systematic mapping study," Software Testing, Verification and Reliability, vol. 33, no. 4, 2023. Available: https://onlinelibrary.wiley.com/doi/10.1002/stvr.1845
[18] Leu, Benjamin, Jonas Volken, Martin Kropp, Nejdet Dogru, Craig Anslow, and Robert Biddle, "Reducing Workload in Using AI-based API REST Test Generation," Proceedings of the 5th ACM/IEEE International Conference on Automation of Software Test (AST), 2024. Available: https://dl.acm.org/doi/10.1145/3644032.3644449
[19] Hossain, Soneya Binta, Antonio Filieri, Matthew B. Dwyer, Sebastian Elbaum, and Willem Visser, "Neural-based test oracle generation: A large-scale evaluation and lessons learned," Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), pp. 120-132, 2023. Available: https://dl.acm.org/doi/10.1145/3611643.3616265
[20] Khan, Shehroz, Abdullah Mughees, Gaadha Sudheerbabu, Tanwir Ahmad, and Dragos Truscan, "Multi-Agent LLM-based Metamorphic Testing for REST APIs," arXiv preprint arXiv:2605.28321, 2026. Available: https://arxiv.org/abs/2605.28321