REFINE: REst From code and INferred Enhancement

André Mesquita Rincon
Department of Computing
Federal University of São Carlos
São Carlos, Brazil
andre.rincon@estudante.ufscar.br

Auri Marcelo Rizzo Vincenzi
INESC TEC, Faculty of Engineering
University of Porto
Porto, Portugal
auri@fe.up.pt

Abstract

Context: OpenAPI Specifications (OAS) can drift from the REST API implementations they describe, declaring status codes the API never returns, omitting existing endpoints, or providing no realistic examples. Every tool that consumes the contract – test generators, SDK generators, mock servers – inherits these defects. **Objective:** This paper presents REFINE (**RE**st **F**rom code and **IN**ferred **E**nhancement), an open-source LLM-based multi-agent tool that automatically aligns an OAS document with the API source code and enriches it with domain-grounded examples. **Method:** REFINE operates in two modes: *specification only* enriches the document with LLM-inferred examples; *specification plus source code* cross-references the document against the source code, producing an enriched specification and a structured discrepancy report. **Results:** Across four open-source APIs, REFINE reconciled between 6 and 27 documentation–implementation discrepancies per project, discovered five missing endpoints in `restcountries` (raising coverage from 44 to 72 status-code pairs), and doubled the documented (endpoint, status-code) pairs in two cases. **Conclusion:** The alignment and enrichment produced by REFINE materially improve automated REST testing effectiveness for both search-based and LLM-based generators, with the latter benefiting additionally from the realistic, domain-grounded examples.
Demo video: https://youtu.be/__A0x1JE8Y8

Keywords

OpenAPI Specification, REST API, Multi-Agent System, LLM

1 Introduction

The OpenAPI Specification (OAS) is the *de facto* contract for REST APIs [15]: most automated tools that consume a REST service – test generators, client SDK generators, mock servers, documentation renderers – read this contract first and treat it as ground truth. In return, the entire ecosystem depends on the document's accuracy. In practice, it rarely is. Developers add endpoints without updating the document; response codes are copied from templates and never validated; parameter descriptions remain placeholders; and examples are absent. The result is a contract that diverges from the implementation it is supposed to describe.

The cost of this drift is concrete and measurable. Consider two representative categories of REST API testing tools: search-based generators, which explore the input space algorithmically to produce JUnit/RestAssured suites, and LLM-based generators, which leverage large language models to synthesize test cases from the specification. Both classes rely on the OAS as their primary oracle, and both inherit whatever defects it contains. Recent empirical work by the authors on the `restcountries` API illustrates the resulting failure modes. EvoMaster [2], a widely used search-based REST API test generator, produced up to 11 failing tests per seed when fed the original specification, each caused by a 404 response that the API never returns. ARTIST [12], an LLM-based multi-agent test generator that operates directly from an OpenAPI document, quarantined a median of 53.5% of its generated tests as `@Ignore` – a JUnit annotation used to mark tests that cannot be executed or validated due to inconsistencies between the expected behavior encoded in the specification and the actual API response. The two outcomes look different (failures versus quarantined tests), but they share a common root cause: in general, tools that consume the contract inherit its defects.

This paper presents REFINE[1], an open-source LLM-based multi-agent tool that addresses the problem at its source. REFINE is designed to be useful independently of any particular downstream tool: given an OpenAPI document and the API source code, it produces (i) an enriched specification populated with realistic examples, with phantom status codes removed and missing ones added; and (ii) a structured Markdown discrepancy report enumerating every divergence between the document and the implementation, ready for human review. Any tool that consumes an OpenAPI document – test generators, SDK generators, mock servers, or documentation renderers – can benefit from a more accurate and complete contract.

REFINE exploits a capability that no rule-based or static-analysis approach offers: an LLM can read a parameter named `name` on an endpoint titled *REST countries API*[2] and propose `"Germany"` or `"Côte d'Ivoire"` as example values, while a rule-based generator might propose `"string"` or `"abc"`. This domain-aware inference turns a syntactic enrichment of the document into a contract that downstream test generators can actually exercise.

The remainder of the paper is organized as follows. Section 2 provides background; Section 3 presents the motivating scenario; Section 4 describes REFINE's architecture; Section 5 highlights its features; Section 6 illustrates its use on `restcountries`; Section 7 discusses implications; Section 8 reviews related work; and Section 9 concludes.

2 Background

Software testing dynamically verifies whether a program behaves as expected over a finite set of test cases, and is conventionally organized into unit, integration, and system testing [3]. API integration testing concentrates on the middle layer, validating interactions between system components through the interfaces they expose and verifying that requests and responses conform to the API contract, checking data exchange, error handling, and faithfulness to the documented behavior [6].

[1] https://github.com/aurimvr/refine
[2] https://github.com/WebFuzzing/Dataset/tree/master/jdk_8_maven/cs/rest/original/restcountries

A REST API follows the architectural principles of Representational State Transfer, a style for building distributed, hypermedia-based systems [5], and is today the dominant approach for exposing functionality in modern systems [8]. Resources identified by URLs are manipulated through HTTP verbs (GET, POST, PUT, DELETE), and each response carries a status code that signals its outcome: 2xx for success, 4xx for client errors, and 5xx for server-side failures [11]. Correctness assertions in automated API tests are typically phrased in terms of which status code each input should trigger, and this oracle is encoded in the specification.

The OpenAPI Specification (OAS) is the de facto standard for documenting REST APIs, capturing paths, methods, parameters, schemas, status codes, and example values [15]. It serves simultaneously as developer documentation, a machine-readable contract, and the primary input for automated tools such as test generators, SDK generators, mock servers, and documentation renderers. Empirical studies show that the quality of this document directly bounds the effectiveness of every tool that consumes it [16, 18]: divergences in the implementation propagate into every downstream artifact, causing false test failures, missing coverage, and incorrect client code – which motivates REFINE.

Large Language Models (LLMs) are AI models trained on massive corpora of text and code, capable of reasoning over both natural-language and programmatic content, with growing applications in software engineering [1, 7, 17]. Two capabilities are particularly relevant to REFINE: LLMs can recognize endpoint definitions across languages and frameworks without per-language rules, keeping REFINE's source-code analyzer language-agnostic; and their domain-language coverage lets them propose plausible example values, e.g., a parameter named `alpha2Code` in a country-data API is far more likely to receive "DE" from an LLM than from a generic random-string generator.

3 Motivating Scenario

This section illustrates how specification drift impairs REST API testing in practice, drawn from an ongoing investigation into LLM-based REST API test generation. We describe the observed failure modes first, then explain how they motivated REFINE's design.

The investigation began with prompt engineering for LLM-generated JUnit/RestAssured suites within the EvoMaster [2] framework, compared against its search-based baseline [14], and later extended with an incremental strategy guided by uncovered code regions [13]. Two limitations emerged: embedding the generator inside EvoMaster tied the setup to that framework's assumptions, and the OpenAPI document's own quality—phantom status codes, missing examples, inconsistent schemas—penalized the generator regardless of prompt quality.

These observations motivated a second cycle of standalone, framework-independent tools: ARTIST [12], an LLM-based multi-agent test generator operating at the specification level; REFINE, the specification-enrichment tool presented here; and COVERTEST[3], under development, to operationalize the incremental coverage-driven strategy. Among these, REFINE targets the limiting factor observed in both prior studies: the quality of the OAS.

[3]https://github.com/aurimrv/COVERTEST

The `restcountries` API illustrates why this matters. Its original specification lists 22 endpoints and 44 status codes. When ARTIST processes it, a median of 53.5% (ARTIST-SPEC) and 24.7% (ARTIST-SPEC-JAR) of generated tests are quarantined as `@Ignore` across 10 seeds, because the specification declares status codes the API never returns (e.g., 404 for endpoints that return 200 with empty bodies) and omits five endpoint–method pairs that the source code implements, making them invisible to any generator that reads the document as ground truth.

This is the defining failure mode of specification-driven testing: when the contract is wrong, every oracle derived from it is wrong by construction. This is not exclusive to ARTIST—the same incorrect status codes caused EvoMaster to produce up to 11 failing tests per seed on the same subject (Section 1), so an enriched, accurate OAS benefits any consumer of the document. REFINE addresses this by removing phantom status codes, adding the missing endpoints (yielding 27 endpoints and 72 status-code pairs), and populating every operation with realistic, domain-grounded examples. Under the enriched specification, ARTIST's `@Ignore` rate drops to 17.1% (ARTIST-SPEC) and 12.6% (ARTIST-SPEC-JAR), with active tests rising from 20 and 67.5 to 107 and 160.5, respectively [12].

These observations motivate two design decisions elaborated next: cross-referencing the specification against the source code rather than rewriting it from scratch, and producing a structured discrepancy report so every correction remains auditable. Because REFINE treats the source code as the reference, divergences may in principle indicate implementation errors rather than documentation errors; the discrepancy report is therefore meant to support, not replace, human judgment.

4 REFINE: Architecture and Design

Figure 1 presents REFINE 's architecture as four sequential layers: Inputs, Multi-Agent Pipeline, External Services, and Outputs. They are coordinated by an `OrchestratorAgent`. The pipeline contains six task agents and runs in one of two modes: *specification-only*, which omits the source-analysis branch, and *specification-plus-source-code*, which executes the full pipeline.

4.1 Inputs

The CLI (`main.py`) accepts two inputs: `–api-spec` (mandatory: a JSON or YAML OpenAPI document, Swagger 2.0 or OAS 3.x) and `–api-src` (optional: the API source code directory). A third flag, `–llm-model`, overrides the model declared in `.env`. The `OrchestratorAgent` validates the paths, detects the OAS version from the document content, and selects the appropriate pipeline mode. The API under analysis never needs to be running. All processing is performed statically.

4.2 Multi-Agent Pipeline

`SpecParserAgent` loads the document and emits a list of typed `EndpointInfo` records, each carrying the path, method, parameters, request body, and response codes. It detects the OAS version from the document itself (`swagger: "2.0"` vs. `openapi: "3.x"`) so that all downstream agents apply version-appropriate transformations.

`SourceAnalyzerAgent` (spec+source-code mode only) walks the source tree under `–api-src`, filters to back-end files by extension.

For each retained file, it prompts the LLM to extract route definitions, HTTP verbs, and the status codes that each handler can return, and then emits a list of `ImplementedEndpoint` records. The agent is language- and framework-agnostic: by delegating extraction to the LLM, it handles Spring, JAX-RS, Flask, FastAPI, Express, and other frameworks without per-framework pattern matching. Explicit prompt rules instruct the model to extract only route definitions, not client-side HTTP calls or URL strings found in documentation.

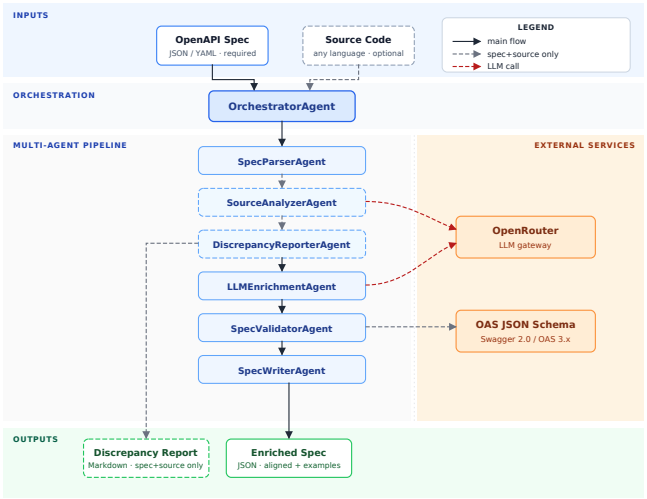

Figure 1: REFINE architecture. Dashed boxes and arrows are active only in *spec+source* mode.

`DiscrepancyReporterAgent` consumes the parsed specification and the implemented endpoints and classifies every endpoint into one of five categories: MATCH, MISSING_IN_IMPL, MISSING_IN_SPEC, RETCODE_MISMATCH, or PARAM_MISMATCH. The findings are serialized as a Markdown report (one section per category, with the endpoint signature, declared codes, observed codes, and a free-text note for each entry) and passed in-memory to the enrichment agent, which uses them to decide which corrections to apply.

`LLMEnrichmentAgent` processes each endpoint in turn. For every documented response code, it prompts the LLM to generate multiple named examples: at least two for 2xx codes (`typical` and `edge_case`) and at least three for 4xx/5xx codes (`invalid_type`, `boundary_violation`, and `null_or_missing`). The system prompt is version-aware: under OAS 3.x, request bodies are emitted as `requestBody`; under Swagger 2.0, the agent emits body examples via `x-parameter-examples` to avoid generating fields the schema forbids. Retries use exponential backoff with explicit non-retriable handling for 401/403 responses.

`SpecValidatorAgent` validates the enriched document against the official JSON Schema for the detected OAS version and applies a catalog of ten auto-repair rules before re-validating. Examples include: converting `in: body` parameters to `requestBody` (R1, OAS 3.x); relocating the `example` field on parameters to the `x-example` vendor extension (R6, Swagger 2.0, where the field is invalid); inserting `required: true` on every path parameter (R5, both versions); normalizing integer response-code keys to strings (R4); and inserting a default `consumes: ["application/x-www-form-urlencoded"]` on operations that use `formData` (R10).

`SpecWriterAgent` deep-copies the original specification, inserts the endpoints discovered only in the implementation (MISSING_IN_SPEC), merges the enrichment payloads, and writes the result to disk as `<spec_name>_<timestamp>.json` alongside the original.

4.3 External Services and Outputs

All LLM calls are routed through the OpenRouter API, enabling the practitioner to select any provider model (`moonshotai/kimi-k2-0905`, `google/gemini-2.5-pro-preview`, etc.) via a single `.env` key. A configurable `RateLimiter` enforces a request-per-minute budget. Token usage and dollar cost are logged to a CSV file for auditability.

Each run produces two artifacts: the enriched specification (always) and a Markdown discrepancy report (available only in specification-plus-source-code mode). Both are saved in the directory of the original specification.

5 REFINE Features

Listing 1 shows REFINE's two typical invocations. The remaining features distinguish REFINE from rule-based or template-driven specification tools.

```
# Spec-only enrichment (no implementation needed)
python main.py --api-spec restcountries.json

# Spec + source: discrepancy report + aligned enrichment
python main.py --api-spec restcountries.json \
  --api-src ../projects/restcountries/src/
```

Listing 1: REFINE invocations.

Domain-aware example generation. Examples are inferred by the LLM from the endpoint's context (operation name, parameter names, descriptions, and the API title) rather than synthesized from generic templates. On a parameter named `alpha2Code` in a country-data API, this produces values like `"DE"` and `"PT"`; on `name` for the same API, values like `"Germany"` and `"Côte d'Ivoire"`. These examples are embedded both in the response `examples` field (where downstream test generators read response payloads) and in an `x-parameter-examples` extension on the operation (where generators read positive and negative parameter values).

Implementation-aligned correction. In spec plus source mode, REFINE treats the source code as the ground truth for endpoints and status codes. Endpoints found only in the implementation are inserted into the enriched specification; status codes documented but never returned are removed; status codes observed in the source but missing from the document are added. This converts the enriched specification from a syntactic enrichment of the original into a contract that actually reflects what the API does.

Discrepancy report. The report groups every finding by category and lists, for each endpoint, the spec-declared codes, the implementation-observed codes, and a textual note. It is intended both as audit evidence (REFINE did not silently delete documented behaviors; it documented what it changed and why) and as input

to the human review process: each finding is a concrete, actionable item that the API team can resolve by updating either side.

Version-aware structural repair. The 10-rule catalog is split into version-agnostic rules (path parameter `required`, duplicate parameter removal, default `title`) and version-specific rules for Swagger 2.0 (no `example` on parameters, no `originalRef`, explicit `consumes` for `formData`) and OAS 3.x (`in: body` →`requestBody`, integer code keys → strings). This ensures the output passes schema validation in standard tooling such as Swagger Editor[4].

Reproducibility and cost transparency. All LLM calls accept a configurable seed for deterministic sampling; token usage is persisted to a CSV file alongside the outputs, enabling the cost to be computed directly from the provider's current prices. For example, a full REFINE run on the original `restcountries` specification in spec+source mode, using the `moonshotai/kimi-k2-0905` model, consumed 48,247 input tokens and 90,787 output tokens—a cost of approximately $1.12 at OpenRouter's pricing.

6 Usage Example

6.1 Specification Before and After REFINE

The four API projects used in this section were selected from the WebFuzzing Dataset [2] and from the authors' broader empirical study. These APIs are well-known benchmarks already used in several other REST API testing studies, which facilitates comparison with the literature. Because REFINE is a new tool, we deliberately limited the case study to APIs for which we could review discrepancy reports entirely by hand; the feasibility of applying REFINE to larger APIs with hundreds of endpoints or heavily modularized codebases is discussed in Section 7. For each project, we compared the original OpenAPI document against the enriched specification produced by REFINE, measuring the number of endpoint–method pairs, the number of (endpoint, status-code) pairs, the count of documentation–implementation discrepancies, and the token and monetary cost of the run. All discrepancies identified by REFINE were manually inspected to verify that they correspond to actual deviations between the specification and the source code.

Across these projects, REFINE consistently reveals either missing behaviors (undocumented endpoints/status codes) or phantom ones (documented responses that the implementation never returns) and rewrites the specification's contract to match the observable behavior. In `features-service` and `restcountries`, the number of status-code pairs doubles or nearly doubles after refinement, indicating that the original specifications documented only a subset of the responses that the code can produce; in `rest-ncs` and `rest-scs`, the apparent decrease in status-code pairs reflects the removal of responses that are never exercised in the implementation.

The discrepancy counts in Table 1 are derived from the discrepancy reports REFINE generates. The reporter groups conflicts into categories such as MISSING_IN_SPEC (endpoint implemented but undocumented), MISSING_IN_IMPL (endpoint documented but not implemented), and RETCODE_MISMATCH (documented and implemented status codes differ); each affected endpoint contributes at most one discrepancy per category.

For instance, considering the `restcountries`, the original specification lists 22 endpoint–method pairs and 44 status-code pairs. After alignment, the enriched specification contains 27 endpoint–method pairs and 72 status-code pairs, with 27 discrepancies recorded in the report. This count arises from two categories. First, five endpoints are classified as MISSING_IN_SPEC: `POST /contribute`, `GET /v1`, `POST /v1`, `GET /v2`, and `POST /v2` are implemented in the Spring Boot code but absent from the original OpenAPI document, each contributing one discrepancy. Second, 22 documented endpoints fall under RETCODE_MISMATCH, because the set of response codes declared in the specification does not match the set that the handlers actually return; for example, `GET /v1/alpha` declares only `200` and `default`, whereas the implementation returns `200`, `400`, `404`, and `500`, totalizing 27 discrepancies.

Table 1: Specifications before and after REFINE across four API projects.

	Original		Refined			Tokens		
Project	EP	SC	EP	SC	Disc.	Input	Output	Cost
features-service	18	18	18	36	18	42,187	55,496	$0.73
restcountries	22	44	27	72	27	48,247	90,787	$1.12
rest-ncs	6	24	6	11	6	15,482	2,052	$0.08
rest-scs	11	44	11	25	11	31,462	40,079	$0.53

EP = endpoint–method pairs; SC = (endpoint, status-code) pairs; Disc. = discrepancy count.

The remaining three subjects show the same pattern of aggregation with different drift profiles. In `features-service`, all 18 endpoint–method pairs appear in the original specification but are marked as RETCODE_MISMATCH, so every documented endpoint requires response-code reconciliation. In `rest-ncs` and `rest-scs`, each original endpoint contributes one discrepancy of type RETCODE_MISMATCH, and the enriched specifications contain fewer status-code pairs than the originals because REFINE removes responses that no reachable execution path can produce.

Listing 2 shows the original entry for `GET /v2/name/{name}` for the `restcountries`. Listing 3 shows the same endpoint in the REFINE-enriched specification (with verbose payload fields elided for space). The contrast illustrates all three core REFINE interventions in one place: the `default` placeholder is replaced by concrete `404` and `500` responses; every parameter and every response receives a realistic example value (`"Germany"`, `"Berlin"`, `"DE"`, `"Federal Republic of Germany"` – values clearly drawn from the country-data domain rather than from generic placeholders); and a dedicated `x-parameter-examples` extension supplies labeled positive and negative parameter values that a downstream test generator can use directly.

```
"/v2/name/{name}": {
"get": {
"operationId": "v2Name",
"parameters": [
{ "name": "fields", "in": "query",
"schema": {"type": "string"} },
{ "name": "name", "in": "path",
"required": true,
"schema": {"type": "string"} },
{ "name": "fullText", "in": "query",
"schema": {"type": "boolean"} }
],
"responses": {
"200": { "description": "Successful response" },
"default": { "description": "Other responses" }
```

Listing 2: `GET /v2/name/{name}` in the *original* specification.

[4]https://editor.swagger.io/

```
"/v2/name/{name}": {
"get": {
"operationId": "v2Name",
"parameters": [
{ "name": "fields", "in": "query",
"schema": {"type": "string"},
"example": "name;capital;population" },
{ "name": "name", "in": "path",
"required": true,
"schema": {"type": "string"},
"example": "Germany" },
{ "name": "fullText", "in": "query",
"schema": {"type": "boolean"},
"example": false }
],
"responses": {
"200": {
"description": "Successful response",
"content": {"application/json": {"examples": {
"example_200": { "value": [{
"name": "Germany", "alpha2Code": "DE",
"alpha3Code": "DEU", "capital": "Berlin",
"region": "Europe", "callingCodes": ["49"],
"currencies": [{"code":"EUR","name":"Euro"}]
/* ...remaining domain fields... */
}]}
}}}
},
"404": {
"description": "Client error - 404",
"content": {"application/json": {"examples": {
"example_404": { "value": {
"status": 404, "message": "Not Found"
}}
}}}
},
"500": {
"description": "Server error - 500",
"content": {"application/json": {"examples": {
"example_500": { "value": {
"status": 500, "message": "Internal Server Error"
}}
}}}

},
"x-parameter-examples": {
"200": [
{"_name":"typical","name":"Germany",
"fields":"name;capital;population","fullText":false},
{"_name":"edge_case","name":"Cote d'Ivoire",
"fields":"name","fullText":true}
],
"404": [
{"_name":"invalid_type","name":123,"fullText":false},
{"_name":"boundary_violation","name":"","fullText":false}
```

Listing 3: Same endpoint after REFINE enrichment.

The discrepancy report associated with this run records the corresponding evidence in human-readable form, as shown in the abbreviated excerpt in Listing 4. Each finding is an actionable item: a project maintainer can audit it, decide whether to amend the specification or the implementation, and track the resolution.

```
## RETCODE_MISMATCH
- GET /v2/name/{name}
spec codes : [200, default]
impl codes : [200, 404, 500]
note: 'default' replaced by concrete 404 and 500
...
## MISSING_IN_SPEC
- POST /contribute (impl codes: 202, 400)
- GET /v1 (impl codes: 200)
- POST /v1 (impl codes: 405)
- GET /v2 (impl codes: 200)
- POST /v2 (impl codes: 405)
```

Listing 4: Excerpt from the discrepancy report for `restcountries` (abridged).

6.2 Impact on Downstream Test Generation

To illustrate how an enriched specification translates into improved testing activity, we report the impact of REFINE on two REST API test generators with different underlying mechanisms: EvoMaster [2], a search-based generator, and ARTIST [12], an LLM-based generator. Using two generators with distinct approaches provides evidence that the improvement is not specific to a single tool or technique, but rather stems from the higher quality of the enriched specification itself.

Table 2 summarizes the median results across 10 seeds when ARTIST is run against `restcountries` under the original and the REFINE-enriched specification.

Table 2: Median results for `restcountries` across 10 seeds. EP/SC = endpoint / status-code coverage (active tests only).

Spec	Metric	EvoMaster	ARTIST-spec	ARTIST-spec-jar
Original	Total tests	63.0	43.0	89.0
	Active tests	63.0	20.0	67.5
	@Ignore (%)	0.0	53.5	24.7
	EP cov. (%)	100.0	77.3	97.7
	SC cov. (%)	95.5	44.3	90.9
Enriched	Total tests	70.0	129.0	183.5
	Active tests	70.0	107.0	160.5
	@Ignore (%)	0.0	17.1	12.6
	EP cov. (%)	100.0	96.3	96.3
	SC cov. (%)	72.2	68.1	93.1

Three observations stand out. First, the `@Ignore` rates drop sharply (53.5% → 17.1% for ARTIST-spec; 24.7% → 12.6% for ARTIST-spec-jar): the realistic examples and the corrected status codes give the correction cycle real responses to validate against, rather than phantom codes that no fix can recover. Second, the active-test counts more than double: ARTIST-spec-jar grows from 67.5 to 160.5 active tests per seed, surpassing EvoMaster's 70 by a factor of 2.3. Third, the apparent drop in EvoMaster's active SC coverage from 95.5% to 72.2% is not a regression but a measurement-scale change: the enriched denominator includes 72 status-code pairs (up from 44), and EvoMaster's suite now exercises a strictly larger absolute number of codes against a strictly larger contract. Even so, EvoMaster benefits from the enriched specification: total test count increases from 63.0 to 70.0. This confirms that the improvements brought by REFINE extend to search-based tools and are not limited to LLM-based generators. ARTIST-spec-jar, by contrast, absorbs the larger denominator, improving to 93.1% SC coverage in the enriched specification.

7 Implications

The discrepancy report turns a previously implicit quality attribute, the alignment between specification and implementation, into an explicit, trackable engineering activity: each RETCODE_MISMATCH or MISSING_IN_SPEC entry is a concrete unit of work the API team can prioritize and close over time. To our knowledge, no prior tool exposes this artifact in this form.

The `x-parameter-examples` extension deserves particular emphasis. Downstream LLM-based test generators consume it as labeled positive and negative inputs per response code, providing a domain anchor for synthesizing requests: a generator

equipped with `name: ["Germany", ""]` and a `404` example labeled `boundary_violation` immediately produces tests for both the success and the empty-string failure case, whereas a generator relying on parameter type alone would have to guess.

Together, these artifacts add value beyond enriching the specification: the discrepancy report serves as audit evidence and as input to human review, since every automatic decision is documented rather than silent, while `x-parameter-examples` transfers domain knowledge directly into the generator's context—an advantage LLM-based tools exploit far more effectively than search-based approaches, which do not reason over natural-language examples.

Limitations. REFINE reasons exclusively over static text, the specification and the source code, and never issues live HTTP requests. Behaviors observable only by exercising the running API with valid credentials, such as authentication, authorization, or session-dependent flows, therefore fall outside the current scope; supporting them requires a live-execution stage with credential injection, left as immediate future work.

Threats to Validity. The four evaluated APIs are well-known WebFuzzing Dataset benchmarks used in prior REST API testing studies, supporting external validity within this class, but their small size leaves scalability to APIs with hundreds of endpoints or non-Spring-Boot frameworks unconfirmed. LLM-based extraction also risks false positives and negatives on complex or dynamically constructed routes; to mitigate this, all discrepancy reports were manually cross-checked against the source code. Finally, since REFINE treats the source code as the reference, a divergence may in principle indicate an implementation error rather than a documentation error—the discrepancy report is designed to surface such cases for human judgment, not to resolve them automatically.

8 Related Work

Several recent tools assess OAS quality from different angles. Kim et al. [9] propose NLPtoREST, which extracts parameter types, constraints, and dependencies from OAS text, improving branch coverage by 103% and request success by 20% across nine APIs. Kim et al. [10]'s RESTGPT replaces this with GPT-3.5 few-shot prompting, raising rule-extraction precision from 79% to 97%. Both focus on parameter constraints, whereas REFINE addresses a broader class of defects (phantom and missing status codes, missing endpoints, absent examples) and ties each correction to source-code evidence via a structured discrepancy report.

Zheng et al. [19]'s RESTLess injects ChatGPT-generated parameter values into OAS, increasing valid test sequences by 42.4% and identifying 38 vulnerabilities across fourteen services. It shares REFINE's enrichment goal but operates purely over the specification, whereas REFINE treats the source code as ground truth for endpoints and status codes.

Yang et al. [18]'s APIDocBooster augments human-facing documentation quality (informativeness, relevance, faithfulness) rather than the structural correctness of the machine-readable contract that REFINE targets. Deng et al. [4]'s LRASGen generates OAS directly from source code with 99.4% endpoint-method accuracy, occupying an adjacent niche: it produces a specification when none exists, whereas REFINE aligns an existing, drift-prone document with the implementation, preserving developer intent rather than regenerating from scratch.

Beyond specification-level tools, Vilela et al. [16] show how specification defects propagate downstream: only 12 of 34 LLM-generated requests executed successfully, with the rest failing due to malformed URIs and semantic inconsistencies originating in the specification itself—exactly the class of problems REFINE detects and corrects upstream.

To the best of our knowledge, no prior open-source tool combines LLM-driven source-code endpoint extraction, status-code reconciliation, domain-aware example generation, OAS-version-aware structural repair, and a structured discrepancy report in a single pipeline.

9 Conclusion

This paper presented REFINE, an open-source LLM-based multi-agent tool that automatically aligns an OAS with the API source code and enriches it with realistic, domain-grounded examples. Its six-agent pipeline operates in two modes (spec-only and spec+source) and produces an enriched specification and, optionally, a structured Markdown discrepancy report.

On `restcountries`, REFINE replaced the `default` response on every endpoint with concrete codes, discovered five endpoint–method pairs that the original document omitted, and populated every parameter and response with realistic country-data examples. The effect on downstream test generation was a near-tripling of the active-test count (67.5 → 160.5 with ARTIST-SPEC-JAR) and a 12-percentage-point reduction in the `@Ignore` rate. The same intervention eliminated a 25% status-code coverage ceiling on two further subjects in the authors' broader evaluation. Improvements were observed across both the LLM-based generator ARTIST and the search-based generator EvoMaster, supporting the conclusion that the benefits of an enriched specification are not tool-specific.

REFINE is the second member of an emerging ecosystem alongside ARTIST, and the first tool, to our knowledge, that operationalizes specification–implementation alignment as a measurable, auditable engineering activity. Future work includes evaluating REFINE on larger APIs with hundreds of endpoints and on projects with heavily modularized codebases, supporting authenticated APIs through configurable credential injection, quantifying the human effort required to triage discrepancy reports in industrial settings, and incorporating dynamic analysis to evaluate API implementation behavior in response to OAS improvements.

Acknowledgements

This work is partially supported by Brazilian Funding Agencies FAPESP (Grant nº 2019/23160-0 and 2023/00001-9), CAPES (Grant nº 001), and CNPq (Grant nº 406941/2025-4).

Artifact Availability

REFINE's source code and documentation are available on Zenodo at https://doi.org/10.5281/zenodo.21461428 (MIT license). The demonstration video is available at https://doi.org/10.5281/zenodo.20370149 (download) and https://youtu.be/__A0x1JE8Y8 (streaming). Full experimental data is available at https://doi.org/10.5281/zenodo.21461518.

References

[1] Ali Al-Kaswan, Toufique Ahmed, Maliheh Izadi, Anand Ashok Sawant, Premkumar Devanbu, and Arie van Deursen. 2023. Extending Source Code Pre-Trained Language Models to Summarise Decompiled Binaries. In *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE Computer Society, Los Alamitos, CA, USA, 260–271. doi:10.1109/SANER56733.2023.00033

[2] Andrea Arcuri. 2019. RESTful API Automated Test Case Generation with EvoMaster. *ACM Trans. Softw. Eng. Methodol.* 28, 1 (Jan. 2019), 1–37. doi:10.1145/3293455 Place: New York, NY, USA.

[3] M. E. Delamaro, J. C. Maldonado, and M. Jino. 2016. *Introdução ao Teste de Software* (2 ed.). Elsevier, Rio de Janeiro, RJ.

[4] Sida Deng, Rubing Huang, Man Zhang, Chenhui Cui, Dave Towey, and Rongcun Wang. 2026. LRASGen: LLM-based RESTful API Specification Generation. *ACM Trans. Softw. Eng. Methodol.* (April 2026), 40. doi:10.1145/3810241

[5] Roy Thomas Fielding and Richard N. Taylor. 2000. *Architectural Styles and the Design of Network-Based Software Architectures*. PhD Thesis. University of California, Irvine.

[6] Amid Golmohammadi, Man Zhang, and Andrea Arcuri. 2023. Testing RESTful APIs: A Survey. *ACM Trans. Softw. Eng. Methodol.* 33, 1 (Nov. 2023), 1–41. doi:10.1145/3617175

[7] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large Language Models for Software Engineering: A Systematic Literature Review. *ACM Trans. Softw. Eng. Methodol.* 33, 8 (Dec. 2024), 1–79. doi:10.1145/3695988

[8] Stefan Karlsson, Robbert Jongeling, Adnan Čaušević, and Daniel Sundmark. 2024. Exploring behaviours of RESTful APIs in an industrial setting. *Software Quality Journal* 32, 3 (July 2024), 1287–1324. doi:10.1007/s11219-024-09686-0

[9] Myeongsoo Kim, Davide Corradini, Saurabh Sinha, Alessandro Orso, Michele Pasqua, Rachel Tzoref-Brill, and Mariano Ceccato. 2023. Enhancing REST API Testing with NLP Techniques. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. ACM, Seattle, WA, USA, 1–12. doi:10.1145/3597926.3598131

[10] Myeongsoo Kim, Tyler Stennett, Dhruv Shah, Saurabh Sinha, and Alessandro Orso. 2024. Leveraging Large Language Models to Improve REST API Testing. In *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER'24)*. Association for Computing Machinery, New York, NY, USA, 37–41. doi:10.1145/3639476.3639769

[11] L. Richardson, M. Amundsen, and S. Ruby. 2013. *RESTful Web APIs: Services for a Changing World*. O'Reilly Media, Santa Rosa, CA, USA.

[12] André M. Rincon and A. M. R. Vincenzi. 2026. ARTIST: Automated REST Testing Intelligent Specification-based Tool. https://github.com/aurimrv/ARTIST

[13] André M. Rincon and Auri M. R. Vincenzi. 2026. Incremental Multi-Model LLM Strategy for Automated Open-Box Integration Testing of REST APIs. *Empirical Software Engineering Journal* (2026). Submitted. Paper under review.

[14] André Mesquita Rincon, Auri Marcelo Rizzo Vincenzi, and João Pascoal Faria. 2025. LLM Prompt Engineering for Automated White-Box Integration Test Generation in REST APIs. In *2025 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE Press, New York, NY, 21–28. doi:10.1109/ICSTW64639.2025.10962507

[15] Souhaila Serbout, Fabio Di Lauro, and Cesare Pautasso. 2022. Web APIs Structures and Data Models Analysis. In *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*. IEEE, New York, NY, 84–91. doi:10.1109/ICSA-C54293.2022.00059

[16] Ricardo F. Vilela, Stevão Alves de Andrade, Eduardo B. F. Santos, Williamson Silva, and Pedro H. D. Valle. 2025. An Intelligent Agent for Automated Test Generation from OpenAPI Specifications. In *Anais do II Workshop sobre Bots na Engenharia de Software (WBOTS)*. Sociedade Brasileira de Computação, Recife, PE, Brasil, 67–76. doi:10.5753/wbots.2025.15217

[17] Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang. 2024. Software Testing With Large Language Models: Survey, Landscape, and Vision. *IEEE Trans. Softw. Eng.* 50, 4 (Feb. 2024), 911–936. doi:10.1109/TSE.2024.3368208

[18] Chengran Yang, Jiakun Liu, Bowen Xu, Christoph Treude, Yunbo Lyu, Junda He, Ming Li, and David Lo. 2025. APIDocBooster: An Extract-Then-Abstract Framework for Augmenting API Documentation. In *2025 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, New York, NY, 36–47. doi:10.1109/ICSME64153.2025.00014

[19] Tao Zheng, Jiang Shao, Jinqiao Dai, Shuyu Jiang, Xingshu Chen, and Changxiang Shen. 2024. RESTLess: Enhancing State-of-the-Art REST API Fuzzing With LLMs in Cloud Service Computing. *IEEE Transactions on Services Computing* 17, 6 (2024), 4225–4238. doi:10.1109/TSC.2024.3489441