You Can REST Now

Decrop, Alix; Perrouin, Gilles; Papadakis, Mike; Devroey, Xavier; Schobbens, Pierre-Yves

Publication date:
2024

Document Version
Early version, also known as pre-print

You Can REST Now: Automated Specification Inference and Black-Box Testing of RESTful APIs with Large Language Models

Alix Decrop
alix.decrop@unamur.be
NADI, University of Namur
Namur, Belgium

Xavier Devroey
xavier.devroey@unamur.be
NADI, University of Namur
Namur, Belgium

Mike Papadakis
michail.papadakis@uni.lu
SnT, University of Luxembourg
Luxembourg, Luxembourg

Pierre-Yves Schobbens
pierre-yves.schobbens@unamur.be
NADI, University of Namur
Namur, Belgium

Gilles Perrouin
gilles.perrouin@unamur.be
NADI, University of Namur
Namur, Belgium

ABSTRACT

RESTful APIs are popular web services, requiring documentation to ease their comprehension, reusability and testing practices. The OpenAPI Specification (OAS) is a widely adopted and machine-readable format used to document such APIs. However, manually documenting RESTful APIs is a time-consuming and error-prone task, resulting in unavailable, incomplete, or imprecise documentation. As RESTful API testing tools require an OpenAPI specification as input, insufficient or informal documentation hampers testing quality. Recently, Large Language Models (LLMs) have demonstrated exceptional abilities to automate tasks based on their colossal training data. Accordingly, such capabilities could be utilized to assist the documentation and testing process of RESTful APIs. In this paper, we present *RESTSpecIT*, the first automated RESTful API specification inference and black-box testing approach leveraging LLMs. The approach requires minimal user input compared to state-of-the-art RESTful API inference and testing tools; Given an API name and an LLM key, HTTP requests are generated and mutated with data returned by the LLM. By sending the requests to the API endpoint, HTTP responses can be analyzed for inference and testing purposes. RESTSpecIT utilizes an in-context *prompt masking* strategy, requiring no model fine-tuning. Our evaluation demonstrates that RESTSpecIT is capable of: (1) inferring specifications with 85.05% of GET routes and 81.05% of query parameters found on average, (2) discovering undocumented and valid routes and parameters, and (3) uncovering server errors in RESTful APIs. Inferred specifications can also be used as testing tool inputs.

KEYWORDS

RESTful APIs, OpenAPI Specification Inference, Black-Box Testing, Request Mutation, Large Language Models

1 INTRODUCTION

Web Application Programming Interfaces (web APIs) offer many services, such as sports information, currency prices and species data. Modern web APIs adhere to the REpresentational State Transfer (REST) architectural style [22], characterized by a set of design principles. Notably, REST APIs utilize the HTTP protocol to send requests and receive responses containing usage-related data. APIs implementing and extending REST principles are termed as RESTful. Developers and users commonly rely on documentation to understand how to use and test RESTful APIs. The OpenAPI specification (OAS) [24] - previously known as the Swagger specification - is a widely adopted industry standard format for documenting RESTful APIs. OpenAPI specifications are both machine and human-readable, relying on JSON or YAML and containing natural language description fields.

Documenting RESTful APIs is a labour-intensive process and as a result developers frequently skip it. This practice results in non-existing, informal, or incomplete documentation. Researchers have proposed some tools that generate formal documentation in the OpenAPI format, but they require advanced API-related inputs [12, 18, 28, 37, 64, 72]. Hence, automating RESTful API documentation with near to zero API knowledge is hard. Additionally, as RESTful APIs become popular, their reliability is also becoming important. Various testing tools have been - and are being - developed, to find bugs and improve the reliability of the tested APIs. However, such tools require an OpenAPI specification of the tested RESTful API as input. The testing results will consequently depend on the quality of this OpenAPI specification. Incomplete or informal specifications are thus highly detrimental to these tools.

To overcome the above-mentioned obstacles, we present *RESTSpecIT*, the first automated approach leveraging Large Language Models to: (1) infer OpenAPI specifications and (2) test RESTful APIs in a black-box and specification-less setup. We hypothesize that Large Language Models (LLMs), e.g., BERT [17] or GPT [55], *have captured through their training process enough knowledge to support RESTful API specification inference and testing*, with minimal input required by the user. At the same time, developers typically use repeated naming conventions or structures [3] and thus, LLMs can make good guesses of the related HTTP request sections. In some sense, the LLMs mutate requests in a way that "seems valid" (i.e. conforming to the natural naming conventions of the related projects and requests) and therefore, returning values that are likely to conform to the given API.

RESTSpecIT requires minimal user input: Only the name of the API and an LLM key for model requests is needed. The main idea of RESTSpecIT consists of generating and mutating HTTP requests without prior knowledge of the API. To do so, the tool utilizes an in-context *prompt masking* strategy, leveraging the LLM (GPT-3.5 in our evaluation) without fine-tuning needed. The strategy consists of *masking* (i.e. hiding a section) of a valid HTTP request and prompting the model for possible values that can replace the

mask. RESTSpecIT generates *mutated* requests by replacing the mask of the request with values found by the LLM. The mutated requests are sent to the corresponding API server endpoint. By analyzing the HTTP response returned by the server, RESTSpecIT can verify if the mutated requests are valid. If a request is valid, it is decomposed into routes and query parameters, which can be inferred into an OpenAPI specification. The valid request is added into a list of seeds for ensuing mutations.

An important aspect of our approach is that by sending mutated requests to an API server, one can discover and exercise different behaviors of the API. In some sense, the tool acts similarly to a standard API *fuzzer*. Interestingly, this process can uncover server errors by analyzing status codes of HTTP responses, i.e. `5xx` status codes which correspond to server errors. Hence, status codes serve as implicit oracles and the related requests can be reported back to developers for further analysis. To illustrate, RESTSpecIT generated the following request during our analysis: `https://api.datamuse.com/words?sp=apple*&v=fruit`. This is actually a bug-triggering input for the *Datamuse* API [10], that leads to an HTTP response with a `500 - Internal Server Error` status code and the following message: `"There was an error processing your request. It has been logged"`. Similarly, when sending several mutated requests to the *CheapShark* API [13], `500` status codes along with internal server error pages were observed.

To summarize, this paper presents the following contributions:

(1) *RESTSpecIT*, a new approach leveraging LLMs to automatically infer OpenAPI specifications and test RESTful APIs in a black-box and specification-less environment (Section 3).
(2) An empirical evaluation of the effectiveness and efficiency of RESTSpecIT in terms of specification inference and testing usages for 10 benchmark APIs (Section 4).
(3) A publicly available replication package with the implementation and evaluation data [5].

We present the background and related work in Section 2. We then describe RESTSpecIT's architecture and design in Section 3. Section 4 presents our research questions, evaluation protocol, and results, while Section 5 provides an additional discussion. Section 6 describes threats to validity. Finally, Section 7 wraps up the paper.

2 BACKGROUND AND RELATED WORK

2.1 RESTful APIs

REpresentational State Transfer (REST) [22] is an architectural style offering several principles to build web-based applications. These principles include stateless communication on top of the HTTP protocol, using HTTP requests to perform various *CRUD - create, read, update, delete -* operations on data resources identified by URIs. Web Application Programming Interfaces (web APIs) implementing or extending REST are entitled RESTful APIs [57]. HTTP status code interpretations may differ depending on the API. For the *GBIF Species* API [20], a response for an invalid endpoint would contain a `404 - Not Found` status code, which is the standard status code to describe a resource that could not be found. However, for the *Bored* API [66], it would contain a `200 - OK` HTTP status code with the following JSON data in the message body: `{"error":"Endpoint not found"}`. Both APIs adhere to the HTTP protocol and utilize it to indicate an invalid route, yet not in the same manner.

Listing 1 OpenAPI specification excerpt for *An API of Ice and Fire* in the YAML format.

```yaml
openapi: 3.1.0
info:
  title: An API of Ice and Fire
  description: OpenAPI Specification for An API of Ice and Fire.
  version: v1
servers:
  - url: 'https://anapioficeandfire.com/api'
    description: Production Server for An API of Ice and Fire.
paths:
  /characters:
    get:
      description: Lists all characters.
      parameters:
        - name: name
          description: Filter characters with the given name.
          in: query
          required: false
          schema:
            type: string
          examples:
            1:
              value: 'Jon+Snow'
            2:
              value: 'Eddard+Stark'
            3:
              value: 'Tyrion+Lannister'
...
```

RESTful API Documentation. To better understand API usage, one can rely on documentation. The *OpenAPI Specification* (OAS) [24] - previously known as the *Swagger Specification* - is a widely adopted format for describing RESTful APIs. OpenAPI specifications are machine-readable and generally structured as data in the JSON or YAML format. OpenAPI specifications are also human-readable, as some fields can contain natural language descriptions. Moreover, editing tools such as the online *Swagger Editor* [62] can convert OpenAPI specifications given as input into human-readable documents. Listing 1 presents an example of an OpenAPI specification excerpt for *An API of Ice and Fire* [60], in the YAML format. The specification describes data related to the API, such as general information in `info`, the API server in `servers`, and the existing API paths in `paths`. Documenting is important, allowing developers to understand, reuse and test RESTful APIs. In addition, Sohan et al. underlined the effectiveness of documenting API usage examples [63]. However, documenting RESTful API specifications is time-consuming and may be error-prone. When API developers neglect the process, the resulting documentation is either unavailable, incomplete, or informal. As a result, automated and formal specification generation has been explored in the literature. To generate or enhance specifications, existing methods require some kind of documentation [12, 28, 37], an HTTP proxy server [64], crawling the API user interface [71, 72] or exploiting API call examples [18]. Unlike RESTSpecIT, these methods require a certain level of API knowledge from their users to guide specification inference.

Black-Box RESTful API Testing. RESTful API testing is an active research field as witnessed by several surveys [19, 27, 59]. State-of-the-art automated testing tools for RESTful APIs commonly use a black-box approach. They require an OpenAPI specification of the RESTful API under test [2, 4, 7–9, 15, 25, 26, 30, 35, 40, 41, 43, 44, 58,

67, 70]. RESTful API Testing consists of generating pseudo-random HTTP requests based on the OpenAPI specification given as input and analyzing the HTTP responses returned by these requests based on an oracle. RESTSpecIT does not require a specification to test a RESTful API: the specification will be generated along the testing process, providing specification coverage information.

2.2 Large Language Models

Large Language Models (LLMs), initially designed for Natural Language Processing (NLP) tasks, are now widely used for a large variety of other tasks. This is due to their capability to learn intricate patterns and semantic representations from vast textual corpora. One initial and influential LLM is the *Bidirectional Encoder Representations from Transformers* (BERT) [17]. BERT uses the *Encoder-only* transformer architecture, focused solely on processing an input sequence and transforming it into abstract representations called *embeddings*. State-of-the-art Encoder-only LLMs include, e.g., *CodeBERT* [21] and *GraphCodeBERT* [29]. Recently, the AI chatbot *ChatGPT* [49] further popularized LLMs. ChatGPT relies upon the *Generative Pre-trained Transformer* (GPT) [55] model architecture. GPT utilizes the *Decoder-only* transformer architecture, focused on output sequences based on previously learned representations, without an encoder module. State-of-the-art Decoder-only LLMs include, e.g., *CodeGen* [48], *GPT-3.5*, and *GPT-4*. The merged *Encoder-Decoder* transformer architecture comprises state-of-the-art models such as *CodeT5* [69] and *PLBART* [1]. As LLM technologies continue to evolve, models and their related strategies are constantly being improved or created. A survey by Zhao et al. [73] discusses recent advances regarding LLMs. Wang et al. offer a preliminary review of LLMs applications for software testing [68].

In-Context Strategies. LLMs can be specialized for specific tasks with 2 different strategies: *fine-tuning* [55] and *in-context learning* [11, 56]. Fine-tuning involves taking a pre-trained model and modifying its parameters or architecture through additional training on a smaller, domain-specific dataset. In-context learning allows LLMs to dynamically update their comprehension based on given inputs, without having to modify the LLM in itself. Prompt engineering explores efficient in-context learning techniques [68], such as *zero-shot learning*, *few-shot learning*, *chain-of-thought* and *self-consistency*. For the following sections, the GPT-3.5 model is considered along with in-context strategies.

Prompt Masking. An in-context strategy consists in *masking* a section of a model prompt, to verify if the LLM is capable of discovering values that could replace the masked section. In this context, masking refers to a technique used to hide data from a set of data. This data is replaced with a *token*, a generic symbol applied onto the data to hide it. The token applied to the data is called a *mask*. Figure 1 presents different examples of prompt masking. The concept of hiding text in a sentence originates from the *Cloze Procedure* by Taylor in 1953 [65]. Deng et al. [16] explored the use of feeding masked code to an LLM to test deep-learning libraries. Meng et al. [46] used LLMs to guide protocol fuzzing, with a message mutation process via LLM interactions. Moreover, Khanfir et al. [36] applied token masks onto code to obtain mutated code from a CodeBERT variant. However, leveraging LLMs to automatically infer OpenAPI specifications and test RESTful APIs without prior knowledge has not been attempted before.

```
Replace the <WORD> token in the following sentence:
The cat is <WORD> home.

Replace the <PARAM> token in the following code snippet:
result = add(3, <PARAM>)

Replace the <VALUE> token in the following API request:
/characters?name=<VALUE>
```

Figure 1: Different examples of prompt masking.

3 APPROACH

In this section, we present our approach. The main idea of RESTSpecIT is to infer OpenAPI specifications and test RESTful APIs by generating and mutating HTTP requests. The process is accomplished with the assistance of a LLM, GPT-3.5 in our implementation. The key role of the model is to return values for the different mutations applied by RESTSpecIT on HTTP request seeds. As LLMs are prone to *hallucinations* [34], the tool meticulously analyzes, parses, and verifies prompt responses. It requires minimal input: only the name of the API and a valid LLM key for model requests are required. Figure 2 shows the overview of RESTSpecIT.

3.1 User Input and Initialization

To specify the user input, we use a JSON configuration file. The minimal amount of input required by the user is the API name and the LLM key, which can be inserted in the corresponding fields. The configuration files contains additional fields related to the tool execution, e.g., `rate-limit` specifies a time (in seconds) to wait between API requests and `temperature` specifies the randomness of the model. The `readme` file contained in the replication package [5] contains a description of all configuration parameters.

Base Data. RESTSpecIT begins by finding the *essential* data of the API. First, the tool generates an empty OpenAPI specification in a JSON file, based on an OpenAPI template. Then, it queries the LLM via prompts to obtain *base* API data: A short API description, the terms of service URL, the contact URL, the contact email address, the API license, the documentation URL, and the server URL. RESTSpecIT parses the model responses, and populates the OpenAPI specification. The tool ensures that only URLs returning `200 - OK` status codes are added to the specification. If a URL is invalid, the model is re-prompted with the same instruction, however specifying that the previous URL was invalid. If after 3 attempts the URL is still invalid, the corresponding field in the specification will consist of an empty string.

Invalid Request Behavior Detection. APIs can handle invalid requests in different ways, e.g., returning `4xx` or `5xx` status codes, or a `200 - OK` status code with an error message. To identify the error-handling behavior of the API, RESTSpecIT voluntarily forms an invalid request composed of the API server URL with the following: `/invalidRoute?invalidParam=invalidValue`. As it is extremely unlikely that such a request is valid, RESTSpecIT stores the response

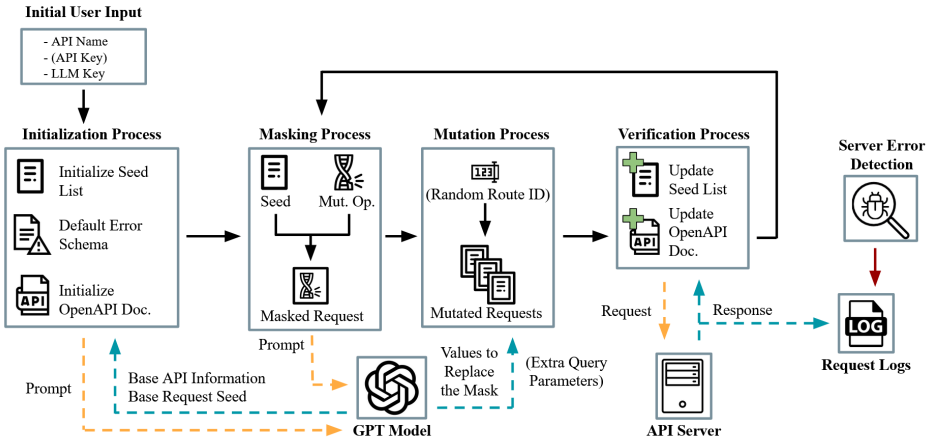

Figure 2: Overview of RESTSpecIT.

error. It will be added with new routes in the OpenAPI specification to describe the response structure of an invalid request.

Initial Seed. To complete initialization, the model is prompted for an example of an HTTP request URL that can be made to the API. If this request is valid, it is inserted into an initialized empty seed list and the data of the request is inferred. If it is not valid, the tool attempts to obtain a valid request 3 additional times. If no valid request is found, the program terminates.

3.2 Masking Process

Seed Selection. As the seed list grows with every new valid request, we select a seed for each mutation. RESTSpecIT can select a seed randomly (`random-seed`) from the seed list, or based on the previously discovered routes (`random-route`, used by default). The latter selects a seed randomly from a filtered sub-list of seeds. This filtered list is generated by (i) selecting a random route from the API routes found by RESTSpecIT. Then, (ii) we add to the filtered list every seed containing the selected route, and (iii) randomly select a seed from this sub-list. This allows all discovered routes to have an equal selection probability.

Mutation Operator Selection. A mutation operator is selected based on the utilized mutation strategy (cf. Section 3.6). Our implementation includes multiple mutation operators to explore different behaviors of the API. Table 1 presents our mutation operators along with application examples. The mutation operators were designed based on request elements that are likely to exercise different behaviors of the API, e.g., exploring a different route or adding a query parameter.

Request Masking. Based on the selected request seed and mutation operator, we mask a section of the request seed. In this context, masking consists in replacing a section of a request with a mutation operator placeholder token. Depending on the mutation operator, 4 different tokens can be applied: `<route>`, `<parameter=value>`, `<parameter>` and `<value>`. The masking process allows the LLM to *guess* token replacements for that request. To avoid mutating potential routes from the API base URL, the configuration file contains a parameter `exclude-routes`. This parameter specifies a list

Table 1: Mutation operators implemented by RESTSpecIT with corresponding examples of masked requests.

Mutation Operator Name	Example
addRoute	/users/25/<route>
removeRoute	/users
modifyRoute	/users/<route>
resetRoutes	/<route>
addParameter	?id=Leo&age=4&<parameter=value>
removeParameter	?id=Leo
modifyParameter	?id=Leo&<parameter=value>
modifyParameterName	?id=Leo&<parameter>=4
modifyParameterValue	?id=Leo&age=<value>
resetParameters	?<parameter=value>

of routes that will not be masked. By default, `exclude-routes` contains the following routes, which are commonly found in API base URLs: `/api`, `/v1`, `/v2`, `/v3`.

3.3 Mutation Process

Model Prompt. The masked request is sent to the model in a crafted prompt. The prompt tells the model to return examples of values that can replace the token in the given request to the API. The prompt is based on a template, customizable for the API and the current mutation operator.

Request Mutation. When the model responds, RESTSpecIT parses the returned data to make sure that the values are of the correct format for the mutation process. If the current mutation operator is related to routes (cf. Table 1), a random integer between 1 and 100 is added as a candidate value for mutations. This is for cases where a route is of the form `/{id}`, which would be easily found with this process. Then, mutated HTTP requests are generated by replacing the masked request token with each value returned by the model.

3.4 Verification Process

Request Validity. Each mutated request generated by the mutation process is sent to the API endpoint, and RESTSpecIT analyzes the API's response. As `2xx` status codes are not sufficient to prove

the validity of a request, the request must satisfy all following conditions:

(1) The response's status code is in the `2xx` - *successful* range.
(2) If the response message contains less than 200 characters, it must not contain the following keywords: `error`, `not found`, `status`, or `invalid`. This allows the detection of errors described in response messages, even with `2xx` status codes. The maximum length of the response data prevents valid data containing the keywords (e.g., as part of a longer text) from making the request invalid, as error responses are frequently short and descriptive.
(3) The type of the response data must not be HTML. Even though HTML data is perfectly valid for web pages, it does not indicate that an API request is valid. First, HTML data could correspond to an in-page error returned by the API server. Second, by analyzing the response data type of over 50 APIs from the *Public APIs GitHub* repository [54], none were HTML.

Error Detection. Mutated requests and their corresponding responses are logged by RESTSpecIT. By analyzing the request logs, the tool is capable of uncovering `5xx` server errors, which can be sent to the relevant API developers for further analysis.

3.5 Inference Process

Documentation Structure. If the mutated request is successfully flagged as valid, we append it to the list of seeds if it does not exist yet. Then, we decompose the request into routes and query parameters. Sections of routes containing an integer value are replaced with a placeholder `/{id}`. For instance, the route `/users/43` would be replaced by `/users/{id}`, as `43` can be replaced with the integer value of any other user. Based on the analyzed data, the OpenAPI specification is enhanced as described below.

Routes. If a route does not exist in the specification, a section is created for it based on the structure of an OpenAPI path described in the official guide [23]. For a valid request using the route, a status code `200 - OK` is specified along with the expected response data. An example of the route response data is added, corresponding to the response of the request that found the route. The invalid route behavior is also specified, as detailed in Section 3.1.

Query Parameters. If the request contains query parameters that do not yet exist in the route structure, they are added to it. The type of the parameter is evaluated and specified, along with its value as a starting example.

Query Parameter Values. Other mutated requests can contain a query parameter that is already described in the inferred specification, however with a value that is not in the parameter examples yet. Thus, the specification accepts up to 10 (maximum for RESTSpecIT) examples of different parameter values.

Human-readable Descriptions. RESTSpecIT can provide LLM-generated descriptions for features found in the OpenAPI specification. Optionally, if `routeDesc` or `parameterDesc` is set to `true` in the configuration file, descriptions will be generated for routes and query parameters, respectively.

Figure 3 presents a simplified mutation process example. As displayed, the `/books` route of a request chosen from the seed list is masked with the `modifyRoute` mutation operator. A prompt is then sent to the model, describing the mask to replace in the request and the expected response structure. The values returned by the model replace the mask and generate new mutated requests. The verification process discovers that the mutated requests containing `book`, `characters` and `house` are valid. However, the mutated request containing `author` is not valid as `author` is not a valid route of the API. The mutated requests containing `characters` and `houses` are added to the seed list, as they are not in the list yet. The request containing `books` is not added as an identical request already exists.

Chosen Request Seed:
`/api/books/4?page=1`

Chosen Mutation Operator:
`modifyRoute`

Masked Request:
`/api/<route>/4?page=1`

Model Prompt:
Return a list containing routes that can replace "`<route>`" in the following request: "`/api/<route>/4?page=1`".

Model Response:
`[books, characters, houses, author]`

Mutated Requests:

`/api/books/4?page=1`	`VALID, EXISTS`
`/api/characters/4?page=1`	`VALID, ADDED`
`/api/houses/4?page=1`	`VALID, ADDED`
`/api/author/4?page=1`	`INVALID, REJECTED`

Figure 3: Simplified example of the mutation process for a request in RESTSpecIT.

3.6 Mutation Strategies

As certain mutation operators can be used in parallel to explore different spaces of APIs, RESTSpecIT implements three different mutation strategies. When it executes a strategy, it sequentially applies each mutation operator associated with the strategy. For each mutation, a new request seed is selected.

Focus Routes. This strategy executes all mutation operators related to routes: `addRoute`, `removeRoute`, `modifyRoute` and `resetRoutes`. The role of the strategy is to explore routes only.

Focus Parameters. This strategy executes all mutation operators related to query parameters: `addParameter`, `removeParameter`, `modifyParameter` and `resetParameters`. The role of the strategy is to explore query parameters only.

Focus All. This strategy executes the *focus routes* and the *focus parameters* strategies, one after the other. This strategy uniformly explores routes and query parameters of the API. Even though RESTSpecIT implements the `modifyParameterName` and `modifyParameterValue` mutation operators, they are not used in any strategy and are only stand-alone.

4 EVALUATION

4.1 Research Questions

We seek to answer the following research questions:

Table 2: RESTful APIs used for the experiment.

API Name	Application Domain	No. GET	No. Par.	Doc. Type	Doc. Used
An API of Ice and Fire	*Game of Thrones* Universe Data	7	17	Plain Text	[61]
Balldontlie	Basketball Data and Statistics	8	12	*Postman* Doc.	[31]
Bored	Random Boredom Activities	1	9	Plain Text	[66]
CheapShark	Game Prices and Deals	4	26	*Postman* Doc.	[13]
CoinCap	Cryptocurrency Data	10	15	*Postman* Doc.	[14]
Datamuse	Word-Finding Query Engine	2	25	Plain Text	[10]
GBIF Species	Species Data Lookup	20	28	Plain Text	[20]
Open Brewery DB	Brewery Data Lookup	6	13	Plain Text	[45]
Random User Generator	Random User Data Generation	1	12	Plain Text	[33]
ReqRes	Mock API for AJAX Requests	4	2	*OpenAPI* Spec.	[32]

RQ.1: *How effective is RESTSpecIT in inferring RESTful API specifications containing routes and parameters in the OpenAPI format?*

RQ.2: *How effective is RESTSpecIT in discovering undocumented and valid routes and parameters of RESTful APIs?*

RQ.3: *How efficient is RESTSpecIT in terms of requests sent, execution time and model costs?*

RQ.4: *How can RESTSpecIT be used for testing RESTful APIs?*

4.2 Experimental Setup

Benchmark. To assess RESTSpecIT, we formed a comprehensive benchmark using a diverse set of RESTful APIs. The selected APIs cover a wide range of application domains. The benchmark comprises APIs referenced on the *Public APIs GitHub* repository [54], APIs suggested by *ChatGPT* [49], and APIs found manually. The APIs all have an online server and do not require an authentication key. Table 2 presents the resulting set of RESTful APIs. In the table, **Doc. Used** specifies the documentation source from where the number of GET routes (**No. GET**) and query parameters (**No. Par.**) were found. **Doc. Type** specifies the format of the documentation: Plain text, an OpenAPI specification [23] or a *Postman* documentation [53]. **No. GET** specifies the number of unique routes of the GET HTTP method. The **No. Par.** metric corresponds to the number of unique query parameters from all GET routes. To illustrate, if an API contains a `/getPet` route accepting the query parameters `[id, name, species]` and a `/getStore` route accepting the query parameters `[id, location]`, the set of query parameters for the API consists of `[id, name, species, location]`.

Procedure. We ran our evaluation using a laptop with a 2.4GHz processor, 16GB of RAM, and a stable *Ethernet* connection. For each API of the benchmark, we apply the following process: RESTSpecIT begins by finding the base data of the API. Then, the *focus all* mutation strategy is applied. When the strategy terminates, the inferred data is extracted and compared with the source API documentation. The process is iteratively applied again. When 2 successive iterations for an API yield no new routes or parameters, the tool considers the API as *fully explored* and it is no longer executed. Listing 2 presents the pseudo-code of the evaluation process.

Listing 2 Algorithm of the evaluation process in Python-like pseudo-code.

```
def runApis(apiList):
    for api in apiList:
        api.strategy = "focusAll"
        api.nbRetries = 2
    remainsExecutions = True
    while remainsExecutions:
        remainsExecutions = False
        for api in apiList:
            if api.nbRetries > 0:
                remainsExecutions = True
                foundData = inferApiData(api)
                if not foundData:
                    api.nbRetries -= 1
                else:
                    api.nbRetries = 2
                    if allFound(foundData):
                        api.nbRetries = 0
                    elif allParametersFound(foundData):
                        api.strategy = "focusRoutes"
                    elif allRoutesFound(foundData):
                        api.strategy = "focusParameters"
```

To reduce the evaluation time, RESTSpecIT did not generate the human-readable descriptions for the routes and parameters in the OpenAPI specification. Moreover, when all routes and parameters of an API are found, no iteration is repeated for the API. If all routes - however, not all parameters - are found, the next API iterations will only apply the *focus parameters* strategy and vice-versa. We repeat the process 10 times to account for randomness in prompt responses and seed selection. Each run starts without the data found in the previous runs. Accordingly, all results are averaged based on these 10 runs.

4.3 RQ.1 - Effectiveness of RESTSpecIT

To evaluate the tool's effectiveness, we considered the following metrics:

% Routes Found. The average (and mean) percentage of documented GET routes found over 10 runs.

% Par. Found. The average (and mean) percentage of documented query parameters found over 10 runs.

FP Par. The average (and mean) amount of false positive query parameters found over 10 runs.

No. RBest / RWorst. The maximum / minimum number of GET routes found in a run from the set of runs.

No. PBest / PWorst. The maximum / minimum number of query parameters found in a run from the set of runs.

Table 3 displays the results obtained for RQ.1. The results demonstrate that our tool is effective at inferring specifications of RESTful APIs, with an average GET route discovery rate of 85.05% and an average query parameter discovery rate of 81.05%. If only considering the best results from all runs, the percentages rise to 91.17% and 89.70%, respectively. Unfortunately, RESTSpecIT can sometimes infer invalid query parameters. On average, 18 query parameters found by RESTSpecIT resulted in false positives for an API. Section 5 explains the difficulty of detecting these false positives.

For the **No. RWorst** column of the Random User Generator API, we observe that the only route of the API was not found. However, the execution specifies that 9/12 query parameters were found, which seems improbable as all query parameters rely on

Table 3: Effectiveness of RESTSpecIT for the APIs of the benchmark. Percentages and FP Par. are structured as *Mean (Standard Deviation)* of the 10 runs and are rounded to 2 decimal places / nearest integer, respectively.

API Name	% Routes Found	% Par. Found	FP Par.	No. RBest	No. PBest	No. RWorst	No. PWorst
An API of Ice and Fire	98.57% (4.29%)	93.53% (8.09%)	31 (5)	7/7	17/17	6/7	13/17
Balldontlie	96.25% (8.00%)	90.83% (4.49%)	36 (12)	8/8	12/12	6/8	10/12
Bored	100.00% (0.00%)	70.00% (22.25%)	1 (1)	1/1	9/9	1/1	4/9
CheapShark	75.00% (0.00%)	66.54% (4.23%)	10 (6)	3/4	19/26	3/4	15/26
CoinCap	77.00% (9.00%)	80.66% (11.72%)	23 (7)	9/10	14/15	6/10	9/15
Datamuse	100.00% (0.00%)	89.6% (4.45%)	5 (2)	1/1	24/25	1/1	20/25
GBIF Species	64.50% (20.30%)	66.79% (5.99%)	31 (11)	16/20	21/28	5/20	16/28
Open Brewery DB	61.67% (7.64%)	79.23% (4.92%)	14 (4)	4/6	11/13	3/6	9/13
Random User Generator	90.00% (30.00%)	73.33% (5.00%)	2 (1)	1/1	9/12	0/1	7/12
ReqRes	87.50% (12.50%)	100.00% (0.00%)	27 (4)	4/4	2/2	3/4	2/2
Total Average	85.05% (9.17%)	81.05% (7.11%)	18 (5)				

the API route. By analyzing the execution results, RESTSpecIT correctly inferred the API specification, but not with the documented API base URL. Indeed, the following API base URL was inferred: `https://api.randomuser.me`. Even though this base URL is not documented in the API, it is valid and functions similarly to the documented `https://randomuser.me/api` endpoint. Thus, query parameters were inferred as the base URL found does not require a `/api` route.

The least amount of routes found for an API was for the Open Brewery DB API, with at most 4/6 routes found. The `/breweries/{id}` route of the API was never found, requiring a specific ID such as `b54b16e1-ac3b-4bff-a11f-f7ae9ddc27e0`. By analyzing the invalid request seeds, RESTSpecIT did attempt to infer the route with integer values for the ID. As integer IDs are not valid in this case, the API rejected the requests, and the route could not be inferred.

For the GBIF Species API, route inference ranges from 5/20 to 16/20, resulting in an average route discovery rate of only 64.50%. This result is due to the chosen seed selection strategy (`random-route`), conferring an equal selection probability to all unique routes inferred. 14 API routes use the `/species/{id}` base, such as `/species/{id}/parents` and `/species/{id}/related`. However, the tool usually finds un-nested routes first such as `/species/match` and `/species/search`. Thus, the probability of selecting the `/species/{id}` route with the `addRoute` mutation operator is lower, causing the path to not be explored for certain executions.

RQ.1 Summary: RESTSpecIT is effective at generating specifications of RESTful APIs in the OpenAPI format, with an average GET route discovery rate of 85.05% and an average query parameter discovery rate of 81.05%.

4.4 RQ.2 - Undocumented API Data

Table 4 presents the undocumented and valid API data found by RESTSpecIT. A total of 3 undocumented GET routes and 6 undocumented query parameters were found. An undocumented and valid API base URL alternative was also found.

For the GBIF Species API, 3 undocumented GET routes were found: `/species/lookup`, `/species/{id}/identifier` and `/species/{id}/metrics`. As requests containing these routes returned `200 - OK` status codes and no error messages in the response data, they were flagged as true positives. Furthermore, we sent an email to the GBIF help desk to obtain feedback regarding the validity of the routes. A data analyst of the API responded, stating that the routes do exist: They are deprecated, but still functional.

For the Bored API, the undocumented query parameters `minparticipants` and `maxparticipants` were found by the tool. Adding each parameter to a valid request causes the `participants` value of the response data to be within range of the specified minimum or maximum participant value. Moreover, by specifying a very large amount of participants, such as `minparticipants=1000`, the response message is `{"error": "No activity found with the specified parameters"}`. This indicates that the API understood the query parameter. Thus, the 2 undocumented query parameters found are true positives.

For the Random User Generator API, the undocumented query parameter `lego` was found by the tool. The parameter is a true positive, as adding it to the query of a valid request to the API causes the response data to contain *lego-related* information. For instance, the `nat` key in the response has a value of `LEGO`, in contrast to a usual nationality value (`CA`, `DE`, `GB`, etc.) when the `lego` parameter is not specified in the request. Moreover, the image values of the `picture` key in the response change to profile pictures of a *lego minifigure*. An alternate base URL was also found for the API, which is stated in RQ1.

For the Datamuse API, we found 3 undocumented query parameters. With a similar verification process, adding the query parameters to a valid request caused changes in the response data. For instance, adding the query parameter `rel_rry` with the value `toto`, responses returned data with words similar to `toto` such as `coco`, `logo`, `kyoto`, etc.

RQ.2 Summary: RESTSpecIT is capable of discovering undocumented and valid data of RESTful APIs. The tool found 3 undocumented GET routes, returning `2xx` status codes without error messages and 6 undocumented query parameters, causing response data changes. An alternate API server URL was also found, having an identical behavior to the base one.

Table 4: Undocumented data found by RESTSpecIT for the APIs of the benchmark.

API Name	Element	Type	Verification
Bored	`minparticipants`	Query Parameter	Response Data Change
Bored	`maxparticipants`	Query Parameter	Response Data Change
Datamuse	`rel_nry`	Query Parameter	Response Data Change
Datamuse	`rel_rhy`	Query Parameter	Response Data Change
Datamuse	`rel_rry`	Query Parameter	Response Data Change
GBIF Species	`/species/lookup`	GET Route	Valid Response, Dev. Feedback
GBIF Species	`/species/{id}/identifier`	GET Route	Valid Response, Dev. Feedback
GBIF Species	`/species/{id}/metrics`	GET Route	Valid Response, Dev. Feedback
Random User Generator	`https://api.randomuser.me`	Server URL	Identical API Behavior
Random User Generator	`lego`	Query Parameter	Response Data Change

4.5 RQ.3 - Efficiency of RESTSpecIT

To evaluate the efficiency of our tool, we considered time, cost and request metrics. The start and end time of each execution was computed, along with the total cost of *input* and *output* tokens for the LLM prompts. A the time of the evaluation, the leveraged GPT-3.5 model had a pricing of $0.001 / 1K tokens for input and $0.002 / 1K tokens for output [51]. Table 5 presents the efficiency results for the 10 runs (from RQ1).

As displayed, execution times range from 264 to 1300 seconds, with an average of 653 seconds per API. Thus, executing RESTSpecIT on the benchmark APIs required less than 11 minutes on average. The number of API requests ranges from 147 to 624, with an average of 288 requests sent per API. This number is fitting to most API request limits. Out of all APIs, the Balldontlie API had the strictest rate limiting (60 requests per minute), which was rarely exceeded; RESTSpecIT does not flood the API servers with requests. Costs of leveraging the LLM range from $0.002 to $0.016, with an average cost of $0.008. Consequently, RESTSpecIT is relatively inexpensive regarding prompt tokens.

In total, the 10 runs consumed 436,649 input tokens and 168,071 output tokens. This represents a cost of $0.44 for input tokens, $0.34 for output tokens, and a total cost of $0.78. This result demonstrates the reasonably cheap way RESTSpecIT leverages the GPT-3.5 model.

> **RQ.3 Summary**: RESTSpecIT is efficient in terms of requests sent, execution time and model costs. The average execution time is 653 seconds for an API. During this time, an average of 288 requests are sent to the API server. The cost of prompting the GPT-3.5 model resulted in $0.008 on average for an API.

4.6 RQ.4 - API Testing with RESTSpecIT

As RESTSpecIT sends mutated requests to API endpoints based on *plausible* mutations found by the leveraged LLM, it exercises the behavior of APIs. Thus, analyzing status codes in API responses could potentially uncover API bugs. Table 6 presents the percentages of status codes obtained for all requests sent throughout the 10 different runs of the evaluation. For each request, the HTTP status code of the response was logged, indicating if the request is valid (`2xx`), invalid due to the client (`4xx`), or invalid due to the server (`5xx`). As displayed, RESTSpecIT generated requests causing server errors for 4 different APIs. The errors were manually replicated after the runs, which proved to be valid server errors and not timeout errors. Moreover, an average of 72.71% requests generated by RESTSpecIT returned `2xx` status codes. However, the Bored API only returns `2xx` status codes. When the API returns an error, it is mentioned in the response data message. By analyzing the Bored API response messages, the results translate to 66.45% (1038 requests) of `2xx` status codes and 33.55% (524 requests) of `4xx` status codes. Consequently, the updated average percentage of `2xx` status codes for APIs is 69.37%.

Additionally, as RESTSpecIT generates machine-readable data (i.e., OpenAPI specifications and API request seeds), we verified if these outputs could be used as inputs of state-of-the-art RESTful API testing tools, relying on OpenAPI specifications. Myeongsoo Kim et al. [38] lists tools such as *Evomaster* [6], *RESTler* [7], *bBOXRT* [39] and *RestTestGen* [67], which all require an OpenAPI specification as input. As RESTSpecIT generated OpenAPI specifications of RESTful APIs that did not exist beforehand, such APIs can now - to a certain extent - be tested with existing tools. The validity and relevance of the generated OpenAPI specifications was verified with the widely used state-of-the-art tool RESTler. RESTler implements a *compile* mode, which takes an OpenAPI specification file given as input and parses it for testing purposes. Then, the *test* mode allows to quickly display the endpoints and methods of the given specification that are covered by RESTler. Table 7 displays the acceptance and coverage of the OpenAPI specifications generated by RESTSpecIT when used in RESTler. As the test mode is capable of uncovering API bugs, we also report the number of `5xx` status codes found by RESTler.

All OpenAPI specifications were correctly compiled (parsed and understood) by RESTler. All routes contained in the specifications were attempted for coverage. However, certain benchmark APIs have more routes covered than the total number of routes referenced in the documentation. This is because RESTSpecIT does not regroup paths with a string as an identifier. For instance, the `/assets/{id}` route of the CoinCap API requires `id` to be replaced by a cryptocurrency string such as `bitcoin`. However, as routes with valid identifiers such as `bitcoin`, `ethereum` and `litecoin` were found by RESTSpecIT, all of these routes are analyzed by RESTler. For 8 out of 10 APIs, all routes were successfully covered with an average successful coverage percentage of 94.5%. Lastly, a `5xx` status code was found for the Datamuse API.

Table 5: Efficiency of RESTSpecIT for the APIs of the benchmark. Whole numbers are structured as *Mean (Standard Deviation)* of the 10 runs and are rounded to the nearest integer. Costs are rounded to 3 decimal places.

API Name	Execution Time (in s.)	No. API Requests	No. Input Model Tokens	No. Output Model Tokens	Input Token Cost (in $)	Output Token Cost (in $)
An API of Ice and Fire	484s (135s)	204 (78)	3110 (1419)	1482 (415)	$0.003	$0.003
Balldontlie	812s (222s)	318 (79)	4895 (1046)	1968 (417)	$0.005	$0.004
Bored	351s (86s)	147 (43)	1910 (549)	851 (224)	$0.002	$0.002
CheapShark	896s (246s)	359 (82)	6499 (2351)	2140 (396)	$0.006	$0.004
CoinCap	794s (283s)	413 (144)	6008 (1870)	2155 (691)	$0.006	$0.004
Datamuse	573s (201s)	213 (51)	2645 (867)	1322 (342)	$0.003	$0.003
GBIF Species	1300s (307s)	624 (155)	9214 (2150)	3553 (801)	$0.009	$0.007
Open Brewery DB	698s (193s)	307 (79)	4899 (1082)	1798 (437)	$0.005	$0.004
Random User Generator	264s (81s)	127 (29)	1432 (298)	697 (138)	$0.001	$0.001
ReqRes	361s (96s)	166 (18)	3052 (245)	841 (127)	$0.003	$0.002

Table 6: Percentages of status codes obtained by RESTSpecIT during the evaluation of the benchmark APIs. Percentages are rounded to 2 decimal places.

API Name	% 2xx Status Codes	% 4xx Status Codes	% 5xx Status Codes	No. Total Requests
An API of Ice and Fire	86.41% (1889)	13.59% (297)	0.00% (0)	2186
Balldontlie	69.87% (2342)	29.53% (990)	**0.60% (20)**	3352
Bored	100.00% (1562)	0.00% (0)	0.00% (0)	1562
CheapShark	49.66% (1849)	32.15% (1197)	**18.18% (677)**	3723
CoinCap	60.61% (2596)	39.39% (1687)	0.00% (0)	4283
Datamuse	84.74% (1927)	14.42% (328)	**0.83% (19)**	2274
GBIF Species	58.02% (3717)	41.98% (2689)	0.00% (0)	6406
Open Brewery DB	57.18% (1836)	42.79% (1374)	**0.03% (1)**	3211
Random User Generator	82.23% (1134)	17.77% (245)	0.00% (0)	1379
ReqRes	78.39% (1429)	21.61% (394)	0.00% (0)	1823

Table 7: Acceptance and coverage of OpenAPI specifications generated by RESTSpecIT when used with RESTler.

API Name	Compiled	% Attempted Route Coverage	% Successful Route Coverage	No. 5xx Codes
An API of Ice and Fire	True	100.00% (7/7)	100.00% (7/7)	0
Balldontlie	True	100.00% (8/8)	100.00% (8/8)	0
Bored	True	100.00% (2/2)	100.00% (2/2)	0
CheapShark	True	100.00% (3/3)	100.00% (3/3)	0
CoinCap	True	100.00% (20/20)	95.00% (19/20)	0
Datamuse	True	100.00% (2/2)	50.00% (1/2)	**1**
GBIF Species	True	100.00% (17/17)	100.00% (17/17)	0
Open Brewery DB	True	100.00% (4/4)	100.00% (4/4)	0
Random User Generator	True	100.00% (1/1)	100.00% (1/1)	0
ReqRes	True	100.00% (21/21)	100.00% (21/21)	0

Table 8: Examples of 3 valid values found by RESTSpecIT for a query parameter from each API of the benchmark.

API Name	Query Parameter	Examples of Valid Values Found
An API of Ice and Fire	`culture`	`Andal, Northmen, Valyrian`
Balldontlie	`search`	`Jordan, Lebron+James, Stephen+Curry`
Bored	`type`	`charity, cooking, social`
CheapShark	`steamAppID`	`115800, 289650, 377160`
CoinCap	`interval`	`d1, h1, m30`
Datamuse	`rel_jjb`	`blue, funny, happy`
GBIF Species	`phylum`	`Arthropoda, Chordata, Mollusca`
Open Brewery DB	`by_city`	`Chicago, New+York, Seattle`
Random User Generator	`nat`	`au, gb, us`
ReqRes	`per_page`	`5, 10, 20`

RQ.4 Summary: RESTSpecIT is useful for testing RESTful APIs. First, RESTSpecIT is itself a testing tool, as it was able to uncover server errors in 4 different APIs through its request mutation process. Second, OpenAPI specifications generated by RESTSpecIT can be successfully used as input of RESTful API testing tools, such as RESTler.

5 DISCUSSION

Query Parameter Value Inference. As a query parameter consists of a key-value pair, it is also interesting to analyze if RESTSpecIT is able to correctly infer parameter values. The validity of a query parameter value can simply consist in a corresponding type, e.g., an integer, a string or a boolean. However, some query parameters may require a more advanced value type, e.g., a character name, a UNIX timestamp, a country code or a species name. As the process of verifying the validity of different parameter value types is a complicated task, Table 8 presents examples of valid values found by RESTSpecIT for a query parameter from each API of the benchmark.

As shown, RESTSpecIT finds correct values for the corresponding query parameters. For instance, the `interval` query parameter of the CoinCap API requires a point-in-time interval as value. The tool found the values `d1`, `h1`, and `m30`, which correspond to point-in-time intervals as described in the CoinCap API documentation [14]. Additionally, the `steamAppID` query parameter of the CheapShark API requires a valid game identifier as value. By using the query parameter with one of the values found by RESTSpecIT (`115800`, `289650`, `377160`) in a request, the response returned a different game data each time. Consequently, the LLM is capable of associating adequate values with API query parameters.

False Positives. Even though RESTSpecIT demonstrated satisfactory results for inferring API specifications, inferred and undocumented query parameters can result in false positives. Given that

REST is not a standard but rather an architectural design, the treatment of invalid query parameters varies depending on the API. However, RESTful APIs tend to ignore invalid query parameters in seemingly valid requests due to the *robustness principle* [47], also known as *Postel's Law*, originating from the *TCP* specification [52]. The principle conveys that *programs receiving messages should accept non-conformant input as long as the meaning is clear*. Consequently, a *meaningful* request containing an invalid query parameter might be treated as valid by the API server, which will ignore the invalid query parameter and only analyze the valid section of the request. Even though false positives impact the documentation generation aspect of the tool, it can be helpful for testing purposes to exercise different behaviors of APIs. Nevertheless, as the leveraged LLM finds query parameters not indicated in the API documentation source, false positives are a side effect of this quality.

Additional OpenAPI Data. RQ1 covered the effectiveness of RESTSpecIT to infer GET routes and query parameters in OpenAPI specifications. However, an OpenAPI specification can contain more data, e.g., API-related URLs (API website, contact page, license), inter-parameter dependencies and response data schemas. As the current focus of the tool is to discover available routes and parameters mostly for testing purposes, such data is not considered in the current work. Moreover, RQ4 results demonstrated that the current data inferred by the tool was sufficient to discover server errors in the RESTful APIs.

API Keys. RESTful APIs sometimes require a key parameter in requests to authenticate API users. While we did not assess APIs requiring a key, RESTSpecIT supports API keys. The key and the API key parameter name can be specified in the tool's configuration file.

Large Language Model Limitations. RESTSpecIT relies on the knowledge that the leveraged LLM has for RESTful APIs. As of December 2023, GPT-3.5 has available training data up to September 2021 [50]. Thus, APIs created after this date might not yield adequate inference results. The LLM might still be capable of *guessing* newer API specifications based on naming conventions and the name of the API. Moreover, our approach also depends on model updates, costs, and response times, which are prone to variations in the future. However, LLMs utilized by RESTSpecIT are interchangeable, only requiring the model API call component to be modified. The remaining components of the tool can still be used, as they only require response strings returned by the LLM component.

6 THREATS TO VALIDITY

Internal Validity. First, the implementation of RESTSpecIT is prone to undetected bugs, leading to potential execution and/or evaluation errors. The tool's code was meticulously reviewed and tested to mitigate this threat. As RESTSpecIT is publicly available in our replication package [5], it is open to reviewers and external users. Similarly, the evaluation process was automated based on output files obtained to avoid errors due to manual steps, and the corresponding results were carefully analyzed thereafter. Second, the number of GET routes and query parameters of the benchmark APIs described in Table 2 might not reflect what is described in the documentation. To mitigate this threat, documentation was carefully explored multiple times to find all documented GET routes and query parameters. Similarly, formal and informal documentation elements were read to discover all available API features. All-in-all, we found inconsistencies in the number of GET routes and query parameters that are described in the documentation and inferred by our tool, i.e., differences between Table 2 and Table 4, which indicate that either the API implementation is incorrect or the documentation is incomplete (or both). We deemed this finding interesting as it points to potential issues that the developers would need to check anyway. Third, the leveraged GPT-3.5 model could have introduced internal validity threats. As the use of the model is entirely black-box-based and zero-shot with a chosen temperature of 0.7, responses could be incorrect w.r.t. the evaluated APIs. We experimented with model prompt templates and in-context strategies beforehand to mitigate this threat. Moreover, RESTSpecIT meticulously parses, analyzes, and validates data after each model prompt to mitigate such errors.

External Validity. First, the RESTful APIs chosen for the benchmark of the evaluation process might not be representative. To mitigate this threat, 10 different RESTful APIs with unique application domains were chosen. The selected APIs varied in terms of structure, such as the number of routes to parameters ratio, static versus dynamic (e.g. `/{id}` placeholder) routes and documentation source format: OpenAPI, Postman or plain text. To recall, all APIs were evaluated with the exact same configuration of the tool. Second, OpenAI server load could have influenced the execution time results when waiting for model prompt responses. To mitigate this threat, we repeated our executions at different times of the day.

7 CONCLUSION AND FUTURE WORK

We presented *RESTSpecIT*, a novel approach leveraging Large Language Models that can automatically infer OpenAPI specifications and test RESTful APIs in a black-box environment. Compared to state-of-the-art API inference and testing tools, RESTSpecIT requires minimal user input: The name of the API and an LLM key for model requests. RESTSpecIT uses a *prompt masking* in-context strategy to retrieve relevant API data from the LLM, requiring no model fine-tuning. With values returned by the model, RESTSpecIT can mutate HTTP requests. By sending mutated requests to the API endpoint and analyzing HTTP responses returned, the OpenAPI specification can be inferred and the API can be tested, by uncovering `5xx` status codes (server errors). Our evaluation results demonstrate that RESTSpecIT can (1) effectively infer API specifications in the OpenAPI format, (2) discover undocumented and valid API data, (3) efficiently be used in terms of requests sent, execution time and model costs, and (4) be used for RESTful API testing. Indeed, RESTSpecIT can uncover server errors in APIs and generate valid OpenAPI specifications that can be used as input of state-of-the-art testing tools.

There is room for future work. First, analyzing query parameters in-depth might reduce or even prevent the inference of false positive query parameters. As adding a query parameter changes the request structure and usually changes the API response, comparing the response data of requests with and without a specific query

parameter could prevent false positives. We will also integrate inter-parameter dependencies in our tool (e.g., [42]). Second, in addition to GET methods on which RESTful APIs mostly rely, we want to support other HTTP methods, such as POST which is used to send data to the server. Third, future work will explore the use of the in-context prompt masking strategy for other application domains. As HTTP requests proved to be effective with RESTSpecIT, masking and mutating other structured data types can be relevant.

REFERENCES

[1] Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified Pre-training for Program Understanding and Generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (Eds.). Association for Computational Linguistics, Online, 2655–2668. https://doi.org/10.18653/v1/2021.naacl-main.211

[2] Seif Ahmed and Abeer Hamdy. 2023. Artificial Bee Colony for Automated Black-Box Testing of RESTful API. In *International Conference on Frontiers of Intelligent Computing: Theory and Applications*. Springer, Springer-Verlag GmbH, Heidelberg, 1–17.

[3] Miltiadis Allamanis, Earl T. Barr, Christian Bird, and Charles Sutton. 2014. Learning natural coding conventions. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), Hong Kong, China, November 16 - 22, 2014*, Shing-Chi Cheung, Alessandro Orso, and Margaret-Anne D. Storey (Eds.). ACM, 281–293. https://doi.org/10.1145/2635868.2635883

[4] Juan C Alonso, Alberto Martin-Lopez, Sergio Segura, Jose Maria Garcia, and Antonio Ruiz-Cortes. 2022. ARTE: Automated Generation of Realistic Test Inputs for Web APIs. *IEEE Transactions on Software Engineering* 49, 1 (2022), 348–363.

[5] Anonymous Anonymous. 2023. *You Can REST Now: Automated Specification Inference and Black-Box Testing of RESTful APIs with Large Language Models - Anonymous Replication Package*. https://doi.org/10.5281/zenodo.10377255

[6] Andrea Arcuri. 2019. RESTful API automated test case generation with EvoMaster. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 28, 1 (2019), 1–37.

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

[8] Vaggelis Atlidakis, Patrice Godefroid, and Marina Polishchuk. 2020. Checking security properties of cloud service REST APIs. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, New York, USA, 387–397.

[9] Ovidiu Baniaș, Diana Florea, Robert Gyalai, and Daniel-Ioan Curiac. 2021. Automated specification-based testing of REST APIs. *Sensors* 21, 16 (2021), 5375.

[10] Doug Beeferman. 2016. Datamuse API. https://www.datamuse.com/api. [Online; Last Accessed 16-November-2023].

[11] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[12] Hanyang Cao, Jean-Rémy Falleri, and Xavier Blanc. 2017. Automated generation of REST API specification from plain HTML documentation. In *Service-Oriented Computing: 15th International Conference, ICSOC 2017, Malaga, Spain, November 13–16, 2017, Proceedings*. Springer, Springer-Verlag GmbH, Heidelberg, 453–461.

[13] CheapShark. 2023. CheapShark API. https://apidocs.cheapshark.com. [Online; Last Accessed 15-November-2023].

[14] CoinCap. 2023. CoinCap API 2.0. https://docs.coincap.io. [Online; Last Accessed 14-November-2023].

[15] Davide Corradini, Michele Pasqua, and Mariano Ceccato. 2023. Automated Black-Box Testing of Mass Assignment Vulnerabilities in RESTful APIs. In *Proceedings of the 45th International Conference on Software Engineering* (Melbourne, Victoria, Australia) *(ICSE '23)*. IEEE Press, New York, USA, 2553–2564. https://doi.org/10.1109/ICSE48619.2023.00213

[16] Yinlin Deng, Chunqiu Steven Xia, Haoran Peng, Chenyuan Yang, and Lingming Zhang. 2023. Large Language Models are Zero-Shot Fuzzers: Fuzzing Deep-Learning Libraries via Large Language Models. In *Proceedings of the 32nd ACM SIGSOFT international symposium on software testing and analysis*. ACM, New York, NY, 423–435.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs.CL]

[18] Hamza Ed-Douibi, Javier Luis Cánovas Izquierdo, and Jordi Cabot. 2017. Example-driven web API specification discovery. In *European Conference on Modelling Foundations and Applications*. Springer, Springer-Verlag GmbH, Heidelberg,, 267–284.

[19] Adeel Ehsan, Mohammed Ahmad ME Abuhaliqa, Cagatay Catal, and Deepti Mishra. 2022. RESTful API testing methodologies: Rationale, challenges, and solution directions. *Applied Sciences* 12, 9 (2022), 4369.

[20] Global Biodiversity Information Facility. 2023. Species API. https://www.gbif.org/developer/species. [Online; Last Accessed 11-November-2023].

[21] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages.

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

[23] Linux Foundation. 2021. OpenAPI Specification. https://swagger.io/specification. [Online; Last Accessed 22-November-2023].

[24] Linux Foundation. 2022. OpenAPI Specification. https://www.openapis.org. [Online; Last Accessed 5-December-2023].

[25] Patrice Godefroid, Bo-Yuan Huang, and Marina Polishchuk. 2020. Intelligent REST API data fuzzing. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, New York, USA, 725–736.

[26] Patrice Godefroid, Daniel Lehmann, and Marina Polishchuk. 2020. Differential regression testing for REST APIs. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM, New York, USA, 312–323.

[27] Amid Golmohammadi, Man Zhang, and Andrea Arcuri. 2023. Testing RESTful APIs: A Survey. *ACM Trans. Softw. Eng. Methodol.* 33, 1, Article 27 (nov 2023), 41 pages. https://doi.org/10.1145/3617175

[28] César González-Mora, Cristina Barros, Irene Garrigós, Jose Zubcoff, Elena Lloret, and Jose-Norberto Mazón. 2023. Improving open data web API documentation through interactivity and natural language generation. *Computer Standards & Interfaces* 83 (2023), 103657.

[29] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, et al. 2020. Graphcodebert: Pre-training code representations with data flow.

[30] Zac Hatfield-Dodds and Dmitry Dygalo. 2022. Deriving semantics-aware fuzzers from web api schemas. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*. ACM/IEEE, New York, USA, 345–346.

[31] hi_im_danny. 2023. Balldontlie API. https://www.balldontlie.io/home.html. [Online; Last Accessed 16-November-2023].

[32] Ben Howdle. 2023. ReqRes API. https://reqres.in/api-docs. [Online; Last Accessed 15-November-2023].

[33] Arron Hunt and Keith Armstrong. 2023. Documentation for the Random User Generator API. https://randomuser.me/documentation. [Online; Last Accessed 11-November-2023].

[34] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *Comput. Surveys* 55, 12 (2023), 1–38.

[35] Stefan Karlsson, Adnan Čaušević, and Daniel Sundmark. 2020. QuickREST: Property-based test generation of OpenAPI-described RESTful APIs. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, New York, USA, 131–141.

[36] Ahmed Khanfir, Renzo Degiovanni, Mike Papadakis, and Yves Le Traon. 2023. Efficient Mutation Testing via Pre-Trained Language Models. *ArXiv* abs/2301.03543 (2023), 1–14. https://api.semanticscholar.org/CorpusID:255545954

[37] 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*. ACM, New York, NY, 1232–1243.

[38] Myeongsoo Kim, Qi Xin, Saurabh Sinha, and Alessandro Orso. 2022. Automated Test Generation for REST APIs: No Time to Rest Yet. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM, New York, USA, 289–301.

[39] Nuno Laranjeiro, João Agnelo, and Jorge Bernardino. 2021. A black box tool for robustness testing of REST services. *IEEE Access* 9 (2021), 24738–24754.

[40] Yi Liu, Yuekang Li, Gelei Deng, Yang Liu, Ruiyuan Wan, Runchao Wu, Dandan Ji, Shiheng Xu, and Minli Bao. 2022. Morest: model-based RESTful API testing with execution feedback. In *Proceedings of the 44th International Conference on Software Engineering*. ACM/IEEE, New York, USA, 1406–1417.

[41] Riyadh Mahmood, Jay Pennington, Danny Tsang, Tan Tran, and Andrea Bogle. 2022. A Framework for Automated API Fuzzing at Enterprise Scale. In *2022 IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, New York, USA, 377–388.

[42] Alberto Martin-Lopez, Sergio Segura, Carlos Müller, and Antonio Ruiz-Cortés. 2022. Specification and Automated Analysis of Inter-Parameter Dependencies in Web APIs. *IEEE Transactions on Services Computing* 15, 4 (2022), 2342–2355.

https://doi.org/10.1109/TSC.2021.3050610
[43] Alberto Martin-Lopez, Sergio Segura, and Antonio Ruiz-Cortés. 2021. RESTest: automated black-box testing of RESTful web APIs. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM, New York, USA, 682–685.
[44] Alberto Martin-Lopez, Sergio Segura, and Antonio Ruiz-Cortés. 2022. Online testing of RESTful APIs: Promises and challenges. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, New York, USA, 408–420.
[45] Chris Mears and Wandering Leaf Studios LLC. 2023. Open Brewery DB Documentation. https://www.openbrewerydb.org/documentation. [Online; Last Accessed 15-November-2023].
[46] Ruijie Meng, Martin Mirchev, Marcel Böhme, and Abhik Roychoudhury. 2024. Large Language Model guided Protocol Fuzzing. In *31st Annual Network and Distributed System Security Symposium (NDSS) 2024*. The Internet Society, Reston, USA.
[47] Lauren Murphy, Tosin Alliyu, Andrew Macvean, Mary Beth Kery, and Brad A Myers. 2017. Preliminary analysis of REST API style guidelines. *Ann Arbor* 1001 (2017), 48109.
[48] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2022. Codegen: An open large language model for code with multi-turn program synthesis.
[49] OpenAI. 2023. ChatGPT. https://chat.openai.com. [Online; Last Accessed 30-November-2023].
[50] OpenAI. 2023. Models. https://platform.openai.com/docs/models. [Online; Last Accessed 15-December-2023].
[51] OpenAI. 2023. Pricing. https://openai.com/pricing. [Online; Last Accessed 13-November-2023].
[52] Jon Postel. 1981. *Transmission control protocol*. Technical Report. Internet Engineering Task Force (IETF).
[53] Postman. 2023. Create API documentation with Postman. https://www.postman.com/api-documentation-tool. [Online; Last Accessed 24-November-2023].
[54] public apis. 2023. Public APIs. https://github.com/public-apis/public-apis. [Online; Last Accessed 14-November-2023].
[55] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training.
[56] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.
[57] Leonard Richardson, Mike Amundsen, and Sam Ruby. 2013. *RESTful Web APIs: Services for a Changing World*. "O'Reilly Media, Inc.", Sebastopol, California.
[58] Sergio Segura, José A Parejo, Javier Troya, and Antonio Ruiz-Cortés. 2018. Metamorphic testing of RESTful web APIs. In *Proceedings of the 40th International Conference on Software Engineering*. IEEE/ACM, New York, USA, 882–882.
[59] Abhinav Sharma, M Revathi, et al. 2018. Automated API testing. In *2018 3rd International Conference on Inventive Computation Technologies (ICICT)*. IEEE, New York, USA, 788–791.
[60] Joakim Skoog. 2023. An API of Ice and Fire. https://anapioficeandfire.com. [Online; Last Accessed 5-December-2023].
[61] Joakim Skoog. 2023. An API of Ice and Fire. https://anapioficeandfire.com/Documentation. [Online; Last Accessed 21-November-2023].
[62] SmartBear Software. 2023. Swagger Editor. https://editor.swagger.io. [Online; Last Accessed 5-December-2023].
[63] SM Sohan, Frank Maurer, Craig Anslow, and Martin P Robillard. 2017. A study of the effectiveness of usage examples in REST API documentation. In *2017 IEEE symposium on visual languages and human-centric computing (VL/HCC)*. IEEE, New York, USA, 53–61.
[64] Sheikh Mohammed Sohan, Craig Anslow, and Frank Maurer. 2015. Spyrest: Automated restful API documentation using an HTTP proxy server (N). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, New York, NY, 271–276.
[65] Wilson L Taylor. 1953. "Cloze procedure": A new tool for measuring readability. *Journalism quarterly* 30, 4 (1953), 415–433.
[66] Drew Thoennes. 2023. Bored API Documentation. https://www.boredapi.com/documentation. [Online; Last Accessed 15-November-2023].
[67] Emanuele Viglianisi, Michael Dallago, and Mariano Ceccato. 2020. RestTestGen: Automated Black-Box Testing of RESTful APIs. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, New York, USA, 142–152.
[68] Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang. 2023. Software Testing with Large Language Model: Survey, Landscape, and Vision. *ArXiv* abs/2307.07221 (2023), 1–20. https://api.semanticscholar.org/CorpusID:259924919
[69] Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. 2021. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.). Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 8696–8708. https://doi.org/10.18653/v1/2021.emnlp-main.685
[70] Huayao Wu, Lixin Xu, Xintao Niu, and Changhai Nie. 2022. Combinatorial testing of restful apis. In *Proceedings of the 44th International Conference on Software Engineering*. ACM, New York, USA, 426–437.
[71] Rahulkrishna Yandrapally, Saurabh Sinha, Rachel Tzoref-Brill, and Ali Mesbah. 2023. Carving UI Tests to Generate API Tests and API Specification. In *Proceedings of the 45th International Conference on Software Engineering* (Melbourne, Victoria, Australia) *(ICSE '23)*. IEEE Press, New York, USA, 1971–1982. https://doi.org/10.1109/ICSE48619.2023.00167
[72] Jinqiu Yang, Erik Wittern, Annie TT Ying, Julian Dolby, and Lin Tan. 2018. Towards Extracting Web API Specifications from Documentation. In *Proceedings of the 15th International Conference on Mining Software Repositories*. ACM, New York, USA, 454–464.
[73] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models.