RESTSpecIT: REST API Specification Generation with LLM-assisted Request Mutations

Alix Decrop*, Xavier Devroey*, Mike Papadakis†, Pierre-Yves Schobbens* and Gilles Perrouin*
*NADI, University of Namur, Namur, Belgium
alix.decrop@unamur.be, xavier.devroey@unamur.be, pierre-yves.schobbens@unamur.be, gilles.perrouin@unamur.be
† SnT, University of Luxembourg, Luxembourg, Luxembourg
michail.papadakis@uni.lu

***Abstract*—In software engineering, documenting is a time-consuming and error-prone task. This is no exception for REST APIs, which are widely used for client-server communications on the web. The OpenAPI Specification (OAS) is an industry standard to document such APIs, and various tools have been developed to assist OAS generation. However, such approaches require advanced inputs such as source code, request samples, or HTML documentation, which are not always available. To address this problem, we present RESTSpecIT, a tool that automatically generates REST API specifications in the OAS format. Our tool takes an API name and endpoint as input and infers relevant documentation. This is done by generating/mutating HTTP requests with the help of an LLM and verifying their validity based on the server responses obtained. Our evaluation demonstrates that RESTSpecIT is effective (94% of routes and 93% of query parameters found on average), efficient (in terms of API requests, execution time, and LLM tokens), and capable of discovering undocumented routes and query parameters.**

The repository of the tool is available at: https://github.com/alixdecr/restspecit.

A demonstration video of the tool is available at: https://youtu.be/OM6Q4Le3kTM.

***Index Terms*—REST APIs, OpenAPI Specification, LLMs, Automation, AI4SE.**

I. Introduction

The REpresentational State Transfer (REST) [1] architectural style is used to build many web systems, with principles such as client-server separation, statelessness, and uniform interfaces through HTTP. Services adhering to such principles are termed REST (or RESTful) APIs. To communicate with REST APIs, clients send HTTP requests containing methods, paths, and optionally query parameters, headers, or payloads (e.g., `GET /currency?id=euro` to retrieve the euro currency, or `POST /users` with a JSON payload `{"name": "john"}` to add a new user named John). Then, API servers respond with information regarding the validity of requests, often containing HTTP status codes and messages (e.g., `200 OK` with the requested data for a success, or `404 Not Found` for an unavailable resource with the message `"User named John does not exist"`).

This is the authors' version. The final version is accepted for publication in the *Proceedings of the IEEE 42nd International Conference on Software Maintenance and Evolution (ICSME 2026)*.

While such APIs follow REST principles, one API may heavily vary from another in terms of usage. For instance, an API specialized in information retrieval might contain many `GET` routes with various query parameters, while an API specialized in user management might contain many `POST` routes with specific payload syntax. Therefore, documentation is crucial for clients to use REST APIs correctly. For this purpose, the OpenAPI Specification (OAS) [2] is a widely adopted standard to formally document REST APIs in a machine-readable format (JSON or YAML).

However, producing documentation remains a challenging task in software engineering [3], and REST APIs are no exception. Indeed, documentation for such APIs may be nonexistent, informal (i.e., not OAS compliant), or incomplete (e.g., lacking routes or parameters). While state-of-the-art approaches exist (cf. Section IV), they often require advanced inputs such as source code, request samples, proxy servers, or HTML documentation, limiting their impact as such data is not always available.

As a result, we present RESTSpecIT, an automated black-box tool that generates REST API specifications in the OAS format. The tool requires minimal input: an API name, endpoint, and access credentials (for LLM and/or API). RESTSpecIT generates and mutates HTTP requests, and analyzes responses to infer and validate the documentation it produces. The tool relies on an LLM, assisting request generation and mutation. We hypothesize that using an LLM is a promising approach for our case as it may (i) recognize the semantics of REST API elements from limited context (e.g., associating `/currency` with a money currency API or identifying `page` as a pagination parameter), (ii) synthesize API information found beyond official documentation (e.g., undocumented routes described in community discussions), and (iii) facilitate description generation for OAS files.

We detail our approach in Section II. Then, we provide evidence of utility through an evaluation in Section III. We detail related work in Section IV. Finally, we provide a conclusion and suggest future work in Section V.

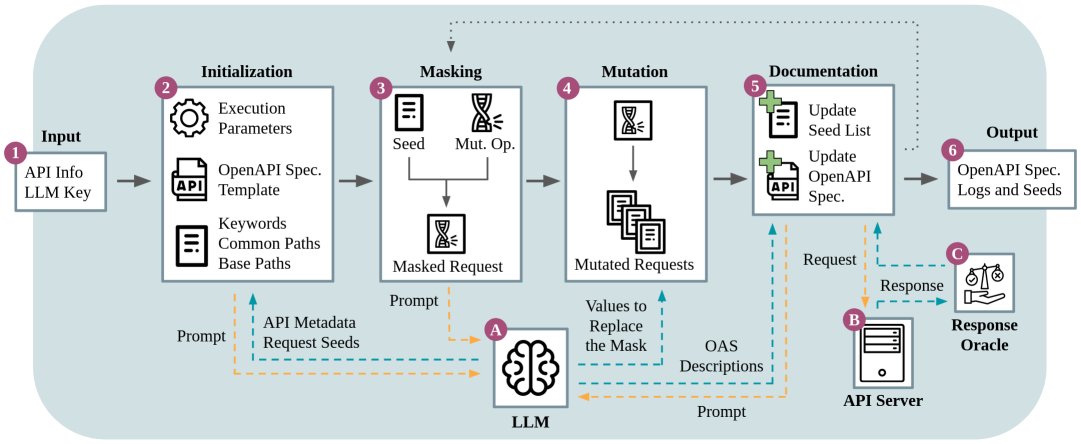

Fig. 1. Architecture of RESTSPECIT.

II. APPROACH

Intended Users and Usage Context. Our tool is aimed at (i) REST API developers who wish to generate or improve their API documentation and (ii) REST API clients who wish to understand APIs with insufficient documentation by generating new documentation with RESTSPECIT.

Provided Artifacts. We provide a GitHub repository [4] containing the implementation of RESTSPECIT, documentation for setup and configuration, our evaluation data for replicability, and the license for availability and reusability.

Architecture. Figure 1 illustrates the architecture of RESTSPECIT. Phases and components are referenced by purple circles (X). First, input is provided by the user in a configuration file (1). To operate, the tool requires an API name and a valid endpoint URL. If the API requires a key or token for request authentication, it should also be provided. If the user leverages an LLM through an API binding (default for the tool), an access key is most likely required to send prompts. Next, the initialization phase occurs (2). During this phase, the execution is configured based on user parameters. An OpenAPI Specification (OAS) file is generated, based on a predefined and up-to-date template for OAS v3.2.0 (latest version as of July 2026). HTTP request seeds are also generated. To do so, the tool infers requests based on (i) API-related keywords returned by the LLM (A), (ii) a predefined list of common API paths (e.g., `/health`, `/ping`, `/swagger`, etc.), and (iii) API paths inferred by the LLM. Then, request seeds are mutated (3)-(4). The masking phase (3) hides a section of a request seed (i.e., replaces a section with a generic placeholder such as `/currency/<path>` or `/pets?<parameter=value>`) for a specific mutation. This process generates a masked request. The mutation phase (4) takes this masked request and asks the LLM for values that could replace the mask. Mutated requests are then obtained by substituting the mask with values returned by the LLM. The mutation process repeats until it reaches a provided budget (time and/or number of iterations). To verify the validity of generated requests, they are sent to the API server (B) with the `OPTIONS` HTTP method. This method is used to ask a server what operations are supported for a specific path. If an API does not handle `OPTIONS`, fallback methods can be specified in the configuration of the tool. By doing so, all methods related to the path are directly analyzed and inferred if valid (e.g., `GET /pets/1`, `DELETE /pets/1`, etc.). If the method requires a payload (i.e., `POST`, `PUT`, or `PATCH`), it is generated by the LLM and sent alongside the request. After obtaining a response from the API server, an oracle (C) is used to assess request validity by analyzing response data. RESTSPECIT rejects requests that triggered invalid response status codes (i.e., `4xx` client error and `5xx` server error ranges). If a response did not contain a status code from these ranges, the corresponding request is partially accepted, depending on the following two checks:

- Error Message Heuristic: If a status code is valid (`2xx`), RESTSPECIT analyzes the response. If it contains an error message with keywords such as "error" or "not found", the request is flagged as invalid due to a false positive. Indeed, status codes are insufficient to determine the exact validity of a request [5].
- Invalid Query Parameters: Due to the Robustness Principle (i.e., Postel's Law) [6], many REST APIs do not return errors when only query parameters of a request are invalid. This behavior may lead to the documentation of invalid parameters. To solve this problem, the oracle compares the responses of requests with and without query parameters. If the responses are identical (i.e., the parameter did not modify the behavior of the initial request), the request is not inferred. This heuristic may fail in some cases (e.g., a parameter sorting a page and the initial page is already sorted) and was not used for our evaluation. Its improvement is left for future work (e.g., trying different parameter values).

Afterwards, API documentation is inferred (5) based on request flagged as valid by the response oracle. The tool extracts information from request-response pairs to generate OAS fields for paths, HTTP methods, parameters (query and path), payloads, and responses. The tool may also support the

generation of OAS descriptions by providing relevant context to the LLM. Finally, when mutations are finished, output is provided to the user (6), consisting of (i) the generated OAS in a `.json` file, (ii) the execution logs in a `.log` file, and (iii) the request seeds (valid and invalid) in a `.json` file.

Implementation. RESTSPECIT is implemented in Python (v3.12), and requires mainly two external packages: `openai` to send prompts to the LLM, and `requests` to send requests to API servers. To easily handle reproducibility, instructions to create a virtual environment (with the exact versions of the packages) are provided in the tool's repository. Users may modify the `config.json` file to configure API information and execution parameters. The LLM in use is substitutable; For GPT and DeepSeek models, users need to insert the model identifier into the `llm-id` configuration parameter, and provide an API key in a `.env` file as `LLM_API_KEY`. It is also possible to add new model families by extending the `BaseModel` class and modifying the `prompt_model` function (serves as a binding for prompts). Additional instructions regarding configuration parameters and LLM substitution are provided in a `README.md` file. Classes and functions of the tool are documented through the Google Python docstring style, facilitating the development of forks and extensions.

Practical Example. Once the tool is correctly set up on a machine (with Python library requirements and LLM access), we can infer API documentation. For instance, we would like to generate the documentation for "An API of Ice and Fire". To do so, we specify its name and endpoint in the `config.json` file. Other parameters may be configured; Their uses are detailed in the tool's repository. Then, we execute the tool and let it run. When the execution terminates, an `outputs/<api-name>` folder is created, containing execution logs, request seeds, and the generated OAS file. Figure 2 illustrates an example of such documentation rendered in an OAS viewer. As shown, routes and parameters that were discovered during execution are documented and described.

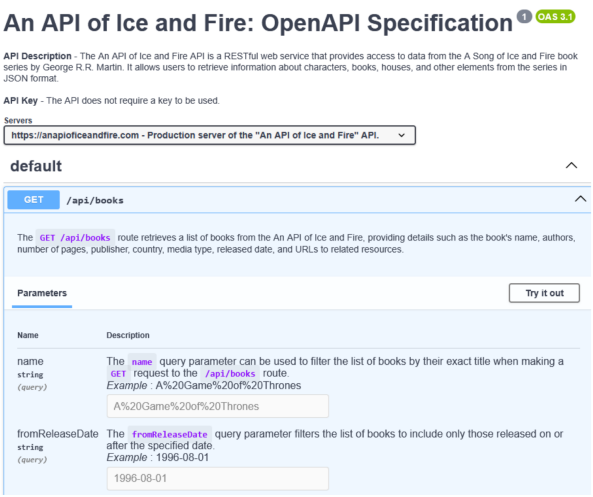

Fig. 2. Example of an OpenAPI Specification generated by RESTSPECIT.

III. EVALUATION

For our evaluation, we formulate the following research question: *How effective and efficient is RESTSpecIT in generating REST API documentation?* By answering it, we aim to assess:

- Effectiveness: How complete the generated OAS files are in terms of API routes (HTTP method + paths) and query parameters w.r.t. given documentation baselines.
- Efficiency: How many requests are sent to API servers, how fast the tool infers documentation, and how many tokens are consumed for LLM-related costs.

Benchmark. We selected five APIs from the Public REST API Benchmark (PRAB), a state-of-the-art benchmark [7] containing the OAS files and structural characteristics of various REST APIs.

Experimental Setup. We ran our evaluation using a laptop with a 2.4GHz processor, 16GB of RAM, and a stable Ethernet connection. We executed our tool on all REST APIs five times each (by specifying their names and endpoints in the configuration) to account for randomness in prompt responses and seed selection. Each execution uses a budget of 10 mutation iterations (cf. Section II). Accordingly, we compute the mean and standard deviation for these executions. Regarding LLM selection, we opted for `deepseek-v4-flash` (with a default temperature of 1), from the latest series of DeepSeek models as of July 2026. We extracted API routes and query parameters from the benchmark documentation (ground truth baseline for the evaluation). The presented metrics were computed using the following definitions:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Where:

- TP (True Positives) correspond to documented elements that were found by our tool.
- FN (False Negatives) correspond to documented elements that were not found by our tool.
- FP (False Positives) correspond to undocumented elements that were found by our tool, yet are invalid.

Effectiveness Results. Table I illustrates the effectiveness results. The results suggest that RESTSPECIT is effective, with 94% of discovered routes and 93% of discovered query parameters on average (i.e., recall). Regarding precision, the tool did not infer false positives for routes in all APIs. However, query parameter precision averages 83%, suggesting that about 17% of inferred query parameters are false positives. The resulting F1-score averages 97% for routes and 87% for query parameters, indicating that RESTSPECIT remains effective for documenting REST APIs.

Moreover, our tool discovered undocumented and valid API data. For instance, the Random User Generator API documentation does not mention the `/portraits` path, yet it was discovered by our tool. The validity of this path was verified, as sending a request with the `/portraits/women/42.jpg` path returned a picture of a woman. Similarly, the undocumented and valid query parameter `rel_rry` was discovered for the Datamuse API: When adding this parameter with the value "toto", responses returned data with similar words such as "coco", "logo", "kyoto", etc. An advanced analysis of undocumented and valid data is left for future work.

Efficiency Results. Table II illustrates the efficiency results in terms of API requests sent, execution time (in seconds), and LLM tokens consumed (input and output). As shown, a typical execution of the tool with a mutation budget of 10 takes around two minutes, sends 243 API requests, and consumes 4,473 input/1,593 output LLM tokens (costing less than $0.01 for `deepseek-v4-flash`). These results suggest that RESTSPECIT is also efficient for documenting REST APIs.

Summary: RESTSPECIT is effective at discovering REST API documentation, with 94% of routes and 93% of query parameters found on average. The tool is also efficient, with an average of 243 API requests sent, 122 seconds of execution time, and few LLM tokens consumed ($< \$0.01$) during a typical execution.

Further Assessing Generated Documentation. To further evaluate RESTSPECIT, we would like to augment existing API specifications and improve their usefulness for API stakeholders by assessing quality [8]. We will design a mixed-method study following state-of-the-art guidelines [9]. Regarding the quantitative assessment of specification augmentation, we will consider metrics such as the number of new routes or new examples RESTSPECIT generated for a given API compared to its original specification. We will assess the documentation's usefulness qualitatively by asking a diverse pool of over 30 users to perform a specific task (e.g., "I would like to retrieve information for a pet named Romeo") using either the original or the augmented specification. We will record whether the task was successful, the time needed to complete it, and provide participants with an open-ended questionnaire. Questions will focus on the experience of exploiting the (augmented) specification: "How informative are the generated examples?", "Are generated descriptions easy to understand?", etc. Specifically, we will adapt the System Usability Scale (SUS) [10] to formulate our questions. We will use this feedback to refine the LLM-based generation.

IV. Related Work

Practices related to REST API documentation are widely explored in software engineering and may be classified into three distinct categories: generation, enhancement, and creation.

Generation. Most practices focus on automatically generating documentation for existing REST APIs. The targeted format is OAS, as it is machine-readable (useful for testing tools) and widely recognized as an industry standard. To generate a REST API documentation, input related to the API is required. Many approaches rely on an HTML/textual documentation of the API [11]–[15]. Other approaches rely on static analysis and API source code [16]–[20]. Some approaches require specific input, such as API call examples [21], an HTTP proxy server [22], or an API user interface [23].

Enhancement. Rather than directly generating REST API documentation, some practices focus on enhancing existing ones. For instance, Kim et al. [24] present NPLTOREST, a tool that can enhance OAS documents by extracting rules from human-readable descriptions. Gonzalez et al. [25] focus on improving documentation with NLP, by adding human-readable descriptions.

Creation. Other practices focus on creating documentation for new (i.e., unimplemented) REST APIs, based on provided human descriptions of their design/uses. Chauhan et al. [26] describe a REST API development approach that uses LLMs and user-provided descriptions.

LLM Reliance. Among the described practices, many utilize recent LLMs [12]–[14], [16], [17], [20], [24]–[26] and provide effective results. This demonstrates that LLMs are suitable for REST API documentation tasks.

Positioning. RESTSPECIT may be classified into the *generation* category. However, in contrast to other approaches, our tool does not require advanced input related to the REST API. As mentioned in Section II, only the API name and endpoint URL are required, along with access keys.

V. Conclusion and Future Work

In this work, we presented RESTSPECIT, a tool capable of generating REST API documentation in the OpenAPI Specification (OAS) format. RESTSPECIT does so by generating and mutating HTTP requests with the assistance of an LLM. By verifying the validity of requests with a response oracle, API-related information is extracted from request-response pairs and progressively inferred into an OAS document. API paths, HTTP methods, parameters, payloads, and responses may be found with the tool. Our approach is the first of its kind to require minimal input, as solely an API name, endpoint, and access keys (for LLM and/or API usage) are required. Based on a preliminary evaluation, we found that RESTSPECIT is effective (94% of routes and 93% of query parameters found on average), efficient, and capable of discovering undocumented routes and query parameters.

For future work, we plan to handle existing OAS files, improve the tool (e.g., response oracle and false positive detection), and focus on undocumented data found by the tool. As mentioned in Section III, we will conduct a qualitative evaluation with REST API users and testers. We will also evaluate additional APIs, LLMs (e.g., Claude, Gemini), and compare cloud vs. local models.

Disclosure

Generative AI was used for limited rephrasing purposes. The authors reviewed the final content and take full responsibility for it. There are no competing interests to declare.

TABLE I

RECALL, PRECISION, AND F1-SCORE OBTAINED FOR ROUTE AND QUERY PARAMETER DISCOVERY ACROSS REST APIS. REPORTED VALUES CORRESPOND TO THE MEAN (AND STANDARD DEVIATION) ACROSS 5 INDEPENDENT EXECUTIONS.

API Name	No. Routes	Route Recall	Route Precision	Route F1	No. Param.	Param. Recall	Param. Precision	Query Param F1
An API of Ice and Fire	7	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)	17	0.98 (0.03)	0.69 (0.14)	0.81 (0.10)
Datamuse	2	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)	24	0.94 (0.02)	0.90 (0.08)	0.92 (0.04)
Open Brewery DB	6	0.90 (0.09)	1.00 (0.00)	0.95 (0.05)	13	0.91 (0.03)	0.85 (0.12)	0.88 (0.08)
Random User Generator	1	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)	12	0.80 (0.05)	0.86 (0.12)	0.82 (0.05)
REST Countries	12	0.82 (0.07)	1.00 (0.00)	0.90 (0.04)	4	1.00 (0.00)	0.83 (0.17)	0.90 (0.10)
Average	**5.60 (3.93)**	**0.94 (0.07)**	**1.00 (0.00)**	**0.97 (0.04)**	**14.00 (6.54)**	**0.93 (0.07)**	**0.83 (0.07)**	**0.87 (0.04)**

TABLE II

EFFICIENCY RESULTS IN TERMS OF API REQUESTS, EXECUTION TIME, AND LLM TOKENS (INPUT AND OUTPUT). REPORTED VALUES CORRESPOND TO THE MEAN (AND STANDARD DEVIATION) ACROSS 5 INDEPENDENT EXECUTIONS.

API Name	API Requests	Execution Time (s)	LLM Input Tokens	LLM Output Tokens
An API of Ice and Fire	125.00 (11.49)	65.04 (8.47)	3121.20 (295.05)	1347.60 (1156.01)
Datamuse	166.40 (69.28)	74.98 (14.18)	3056.80 (419.42)	1722.60 (403.32)
Open Brewery DB	132.80 (9.23)	97.36 (6.57)	3129.80 (385.16)	1311.00 (112.81)
Random User Generator	298.40 (166.79)	118.55 (49.49)	5286.40 (1942.59)	2035.60 (1084.72)
REST Countries	490.40 (107.25)	254.48 (52.07)	7771.20 (1540.84)	1547.60 (346.96)
Average	**242.60 (138.72)**	**122.08 (68.74)**	**4473.08 (1853.48)**	**1592.88 (266.27)**

ACKNOWLEDGMENTS

Gilles Perrouin is an FNRS Research Associate. This research was partially funded by the CyberExcellence by DigitalWallonia project (No. 2110186), funded by the Public Service of Wallonia (SPW Recherche).

REFERENCES

[1] R. T. Fielding, *Architectural styles and the design of network-based software architectures*. University of California, Irvine, 2000.

[2] SmartBear, "OpenAPI specification," 2026, accessed: 2026-04-23. [Online]. Available: https://swagger.io/specification

[3] E. Aghajani, C. Nagy, O. L. Vega-Márquez, M. Linares-Vásquez, L. Moreno, G. Bavota, and M. Lanza, "Software documentation issues unveiled," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 1199–1210.

[4] A. Decrop, "RESTSpecIT GitHub repository," 2026. [Online]. Available: https://github.com/alixdecr/restspecit

[5] A. Decrop, M. Papadakis, and G. Perrouin, "Analyzing status code misuses in REST API specifications," in *Proceedings of the 26th International Conference on Web Engineering, ICWE 2026*, 2026.

[6] J. Postel, "DoD standard transmission control protocol," RFC 761, jan 1980. [Online]. Available: https://www.rfc-editor.org/info/rfc761

[7] A. Decrop, S. Eraso, X. Devroey, and G. Perrouin, "A public benchmark of REST APIs," in *2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR)*. IEEE, 2025, pp. 421–433.

[8] C. Treude, J. Middleton, and T. Atapattu, "Beyond accuracy: Assessing software documentation quality," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 1509–1512.

[9] M.-A. Storey, R. Hoda, A. Maciel Paz Milani, and M. T. Baldassarre, "Guiding principles for mixed methods research in software engineering," *Empirical Software Engineering*, vol. 30, no. 5, p. 138, 2025.

[10] J. Brooke *et al.*, "SUS-a quick and dirty usability scale," *Usability evaluation in industry*, vol. 189, no. 194, pp. 4–7, 1996.

[11] H. Cao, J.-R. Falleri, and X. Blanc, "Automated generation of REST API specification from plain HTML documentation," in *International Conference on Service-Oriented Computing*. Springer, 2017, pp. 453–461.

[12] A. Kesraoui, O. I. Houssaini, and A. Mourhir, "Design, implementation, and evaluation of an LLM-powered system for automatic API generation according to OpenAPI specification," in *2025 International Conference on Intelligent Systems: Theories and Applications (SITA)*. IEEE, 2025, pp. 1–8.

[13] K. Lazar, M. Vetzler, G. Uziel, D. Boaz, E. Goldbraich, D. Amid, and A. Anaby-Tavor, "SpeCrawler: Generating OpenAPI specifications from API documentation using large language models," *arXiv preprint arXiv:2402.11625*, 2024.

[14] K. Lazar, M. Vetzler, K. Kate, J. Tsay, D. Boaz, H. Gupta, A. Shinnar, R. D. Vallam, D. Amid, E. Goldbraich *et al.*, "Generating OpenAPI specifications from online API documentation with large language models," in *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, 2025, pp. 237–253.

[15] J. Yang, E. Wittern, A. T. Ying, J. Dolby, and L. Tan, "Towards extracting web API specifications from documentation," in *Proceedings of the 15th International Conference on Mining Software Repositories*, 2018, pp. 454–464.

[16] H. Chen, Y. Li, C. Chen, F. Lin, and W. Li, "OpenAI for OpenAPI: Automated generation of REST API specification via LLMs," *arXiv preprint arXiv:2601.12735*, 2026.

[17] S. Deng, R. Huang, M. Zhang, C. Cui, D. Towey, and R. Wang, "LRASGen: LLM-based RESTful API specification generation," *ACM Transactions on Software Engineering and Methodology*, 2025.

[18] R. Huang, M. Motwani, I. Martinez, and A. Orso, "Generating REST API specifications through static analysis," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–13.

[19] A. Lercher, C. Macho, C. Bauer, and M. Pinzger, "Generating accurate OpenAPI descriptions from Java source code," *arXiv preprint arXiv:2410.23873*, 2024.

[20] A. Smardas and K. Kritikos, "Towards the automatic production of OpenAPI specifications from source code," in *2025 12th International Conference on Future Internet of Things and Cloud (FiCloud)*. IEEE, 2025, pp. 342–349.

[21] H. Ed-Douibi, J. L. Cánovas Izquierdo, and J. Cabot, "Example-driven web API specification discovery," in *European Conference on Modelling Foundations and Applications*. Springer, 2017, pp. 267–284.

[22] S. M. Sohan, C. Anslow, and F. Maurer, "SpyREST: Automated RESTful API documentation using an HTTP proxy server," in *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering*, 2015, pp. 271–276.

[23] R. Yandrapally, S. Sinha, R. Tzoref-Brill, and A. Mesbah, "Carving UI tests to generate API tests and API specification," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 1971–1982.

[24] M. Kim, D. Corradini, S. Sinha, A. Orso, M. Pasqua, R. Tzoref-Brill, and M. Ceccato, "Enhancing REST API testing with NLP techniques," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2023, pp. 1232–1243.

[25] C. González-Mora, C. Barros, I. Garrigós, J. Zubcoff, E. Lloret, and J.-N. Mazón, "Improving open data web API documentation through interactivity and natural language generation," *Computer Standards & Interfaces*, vol. 83, p. 103657, 2023.

[26] S. Chauhan, Z. Rasheed, A. M. Sami, Z. Zhang, J. Rasku, K.-K. Kemell, and P. Abrahamsson, "LLM-generated microservice implementations from RESTful API definitions," *arXiv preprint arXiv:2502.09766*, 2025.